



HAL
open science

Bases de données et cartographie en Sciences de la terre avec MariaDB (MySQL) et Python L3-Terre & Environnement

François Métivier

► To cite this version:

François Métivier. Bases de données et cartographie en Sciences de la terre avec MariaDB (MySQL) et Python L3-Terre & Environnement. Licence. Paris, France. 2020. cel-01877606v3

HAL Id: cel-01877606

<https://hal.science/cel-01877606v3>

Submitted on 12 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - ShareAlike 4.0 International License

Bases de données et cartographie en Sciences de la terre avec MariaDB (MySQL) et Python L3-Terre & Environnement

François Métivier

Université de Paris,
Institut de physique du globe de Paris,
1 rue Jussieu, 75005 Paris.

2 septembre 2021

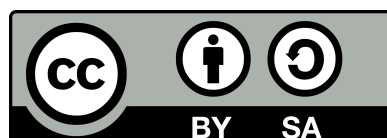


Table des matières

I	Bases de données	11
1	Introduction	13
1.1	Questions (réponses)	13
1.2	Objectifs	13
1.3	Ma première base de données	14
2	Créer et utiliser une base	15
2.1	Objectifs	15
2.2	Créer la base Toto	15
2.3	Créer la table tata	15
2.4	Modifier la structure d'une table	16
2.5	Enregistrer des données	16
2.6	Voir, effacer et mettre à jour	17
2.6.1	Voir	17
2.6.2	Effacer	17
2.6.3	Mettre à jour	17
2.7	Sauvegarder ou transférer une base	17
3	Les requêtes sur une table	21
3.1	La base de données Parcelle	21
3.2	Structure et chargement de la table météorologique : <code>imetos</code>	21
3.3	Requêtes simples	21
3.3.1	Sélectionnez un ou plusieurs champs d'une table	21
3.3.2	Ordonner	23
3.3.3	Distinguer	23
3.3.4	Calculer	23
3.3.5	Exporter	23
3.4	Les conditions	24
3.4.1	Comparaison	24
3.4.2	Conditions multiples : utilisation des conjonctions	24
3.5	L'emploi de fonctions	25
3.5.1	Fonctions concernant le temps	25
3.5.2	Fonctions de calculs	25
3.5.3	Fonctions concernant les chaînes de caractères	25
3.6	L'agrégation de données	25
3.7	Applications	26
3.8	Requêtes emboîtées	27
3.8.1	Principe	27
3.8.2	Applications	27
3.9	Vues	28
3.10	Conclusion	28
4	Tables multiples, clés et liens	29
4.1	La base Parcours	29
4.1.1	Structure de la base	29
4.1.2	Implémentation	30
4.1.3	Insérer les données	31
4.2	Requêtes	31
4.2.1	Patrons de recherche avec REGEXP	31

4.2.2	Requêtes sur plusieurs tables	32
4.2.3	Types de jointures	33
4.2.4	Requêtes simples sur une ou deux tables	34
4.2.5	Requêtes emboîtées	35
5	Utiliser Python et MySQL	37
5.1	Introduction	37
5.2	Exemples	37
5.2.1	Récupérer des données numériques	37
5.2.2	Récupérer des données mixtes	38
5.3	SQL DDL	39
5.3.1	Traiter les erreurs	42
5.3.2	OpenDB : a simple wrapper	42
5.3.3	Précipitations : seconde implémentation	43
5.4	Applications	44
5.4.1	Cycles journaliers de température à Saint-Maur	44
5.4.2	Direction des vents at Saint Maur	45
6	Pandas et SQL	47
6.1	Connexion	47
6.1.1	SQLAlchemy	47
6.1.2	Importer une table dans un dataframe	47
6.1.3	Utiliser des requêtes	48
6.1.4	Charger des dataframes dans une base SQL	48
6.2	Applications	49
6.2.1	Précipitations : seconde implémentation	49
6.2.2	Cycles journaliers de température à Saint-Maur deuxième version	50
6.2.3	Direction des vents at Saint Maur	50
6.3	les capacités de regroupement de Pandas	51
II	Cartographie	53
7	Introduction	55
8	Les fonds de cartes	57
8.1	Première carte	57
8.1.1	Commentaires	58
8.2	Coordonnées, système de référence et projections	58
8.2.1	Sélectionner une projection	58
8.3	Les limites de la carte	59
8.4	Echelle	59
8.5	Les fonds de carte disponibles	64
8.5.1	Données vectorielles	64
8.5.2	Images et fonds topographiques	66
8.5.3	Images de fond	67
8.5.4	les serveurs de tuiles wms et wmts	67
9	Ajouter des données	73
9.1	Points et trajets	73
9.2	Deux exemples en Sciences de la terre	75
9.2.1	Carte de la sismicité	75
9.2.2	USGS Earthquake catalogue	77
9.2.3	Variation temporelle du champs magnétique (contour plot)	78
10	Pyproj	81

11 Les shapefiles	83
11.1 Formes et géométries	83
11.2 OGR Data model	83
11.3 Shapefile structure	83
11.4 Représenter un shapefile	85
11.4.1 Contours	85
11.4.2 Colorier les polygones	86
11.5 Les shapefiles comportant une projection	88
11.6 Écrire des données dans un shapefile	88
12 Les rasters	91
12.1 Représenter un raster	91
12.1.1 Landsat prétraitée de l'Aquitaine	91
12.1.2 Topographie de la Guadeloupe	92
12.2 Les images GeoTiff	95
12.2.1 Histogramme	95
12.2.2 Etirement des valeurs de pixels et fusion des canaux	97
12.2.3 Joindre et découper	98
13 Exemples d'applications	103
13.1 Base des Parcours	103
13.2 Hydrogéologie des lacs des Landes	105
13.3 Lacs des Landes, seconde manière	109
14 Faire des cartes en ligne	113
14.1 Introduction : Leaflet et Folium	113
14.2 Le fond de carte	113
14.3 La gestion des tuiles	114
14.4 Ajouter des données : les séismes des 7 derniers jours	115
14.5 Les clusters	117
14.6 GeoJson	118
14.7 Choroplèthes	121
14.8 Ajouter des Traçés	122
14.8.1 Trajectoires d'ouragans	122
14.8.2 Carte piézométrique	123
A Rappels Python	129
A.1 Introduction	129
A.2 Structure d'un programme	131
A.2.1 Libraries	131
A.2.2 fonctions	131
A.2.3 classes	131
A.3 Exemples	132
A.3.1 Premier graphique	132
A.3.2 Surface libre d'une nappe	132
A.4 L'utilisation des axes	134
A.5 Collections et formes géométriques	136
A.6 Légendes	137
A.7 La fonction zip	139
B Bases de données supplémentaires	141
B.1 La base GEMS	141
B.1.1 Requêtes simples sur une table	142
B.1.2 Requêtes sur une ou plusieurs tables	142
B.1.3 Les requêtes emboîtées	142
B.2 Les Charts	144
B.2.1 Requêtes simples	144
B.2.2 Requêtes emboîtées	144

C Ressources internet	145
C.1 language=SQL	145
C.2 Python	145
C.3 Basemap, cartopy et SIG	145

Table des figures

1.1	Schéma d'un carnet d'adresses	14
3.1	L'observatoire de Saint-Maur en 2007	22
4.1	Schéma de la base Parcours	29
4.2	Parcours : Schéma alternatif	30
4.3	Différents types de jointures en SQL	33
5.1	Nombre d'inscrits en Génie de l'Environnement depuis la création de la base <code>Parcours</code>	38
5.2	Précipitations journalières à Saint-Maur des Fossés	40
5.3	Température journalière, durant l'été, à l'observatoire de Saint-Maur	45
5.4	Direction des vents à l'observatoire de Saint-Maur	46
8.1	Tracé des côtes (projection Plate Carrée)	57
8.2	Projection de Mollweide centrée sur le méridien de Greenwich à gauche et sur 180° à droite.	59
8.3	Projection transverse Mercator. à gauche <code>basemap</code> , à droite <code>cartopy</code>	60
8.4	Grilles en Plate Carrée et Mollweide, la distance entre méridiens est de 40° et entre parallèle de 30°	61
8.5	Personnalisation de grilles en Plate Carrée	62
8.6	Grilles, coordonnées et échelle en Lambert conforme	63
8.7	Fonds de carte disponibles dans <code>cartopy</code>	65
8.8	Fonds de carte disponibles dans <code>basemap</code> et <code>cartopy</code> . À REFAIRE SANS BASEMAP	66
8.9	Image Blue Marble incluse comme fond de carte dans <code>cartopy</code>	67
8.10	Utilisation de serveurs de tuiles avec <code>cartopy</code> . Image Landsat ETM de la France, 16 août 1999	69
8.11	Utilisation de serveurs de tuiles avec <code>cartopy</code> . De haut en bas : Blue Marble, ASTER GDEM, Densité de population 2020.	70
9.1	Ajouter des points dans différentes projections	74
9.2	Deux trajets	75
9.3	Sismicité mondiale (Février 2017)	75
9.4	Sismicité mondiale des trente dernier jours ($M \geq 5$). Exécuté le 19/09/2020	78
9.5	Variation temporelle de la composante verticale du champs magnétique \dot{Z}	79
11.1	Modèle de données OGR	84
11.2	Pays du monde, projection Plate Carrée	86
11.3	Population mondiale (en 2017).	86
12.1	Landsat TM8 des landes d'Aquitaine.	93
12.2	MNT ASTER de la Guadeloupe...en un seul bout! CHANGER OUR LE BON MNT GUAD + MARTI	94
12.3	Bassin de Bayanbulak Landsat 5 144030, Canal 10.	96
12.4	Histogramme des trois canaux (10,20,30) de la Landsat 144030	97
12.5	Bassin de Bayanbulak, Landsat 5 144030; image composée des trois canaux (R :30,V :20,B :10) étirés	98
12.6	Bassin de Bayanbulak Landsat 5 145030, image composée des trois canaux (R :30,V :20,B :10) étirés	99
12.7	Bassin de Bayanbulak. image composée à partir des trois canaux (R :30,V :20,B :10) étirés de deux images Landsat 5 (144030, 145030). Projections plate carrée et azimutale équidistante.	101
13.1	Nombre agrégé d'étudiants par département un an avant leur arrivée à l'UFR STEP	106

13.2 Hydrogéologie des bassins des étangs de Cazaux et de Parentis. Image satellite : ESRI World Imagery. Hydrogéologie : SIGES Aquitaine.	109
13.3 Hydrogéologie des bassins des étangs de Cazaux et de Parentis 2ème version. En haut <code>basemap</code> , en bas <code>cartopy</code>	112
14.1 Position des différents sites de l'IPGP.	114
14.2 Zoom sur une carte composée de trois tuiles superposées : un DEM ASTER en grisé, une carte de densité de population (2020) et le fond OpenStreetMap.	115
14.3 Carte des séismes ($M \geq 1$) des 7 derniers jours. Source base de données de l'USGS	117
14.4 Localité des étudiants un an avant leur arrivée en Licence ST	118
14.5 Population mondiale première version : GeoJson + GeoJsonPopup	121
14.6 Population mondiale seconde version ; Choropleths	122
14.7 Trajectoire des ouragans de 2018.	123
14.8 Carte piézométrique de la nappe des Landes	125
A.1 $x \cos(x)$	132
A.2 Profils d'une nappe en état stationnaire pour différentes conductivités hydrauliques.	133
A.3 Doubler un axe avec <code>axes</code>	135
A.4 Positionnement des repères avec <code>axes</code>	136
A.5 Utilisation des objets <code>collections</code> et <code>patches</code>	138
B.1 Lien entre deux tables	142
B.2 Schéma de la Base GEMS	143

Avant de poursuivre

Ces notes résument le contenu d'un cours pratique de premier cycle (L3) pour les étudiants en sciences de la Terre. L'objectif est de les aider à gérer leurs données et créer des cartes et figures en utilisant deux langages de programmation le SQL et Python 3. Ce cours implique une connaissance de la programmation en Python (l'annexe A résume les connaissances dont nous aurons besoin).

Les choix faits ici sont

1. d'utiliser des logiciels libres et si possible opensource,
2. de forcer les étudiants à programmer un peu,
3. d'appliquer les connaissances acquises aux sciences de la Terre,
4. de progresser par l'exemple en restant pratique.

Ces notes sont accompagnées de matériel permettant de réaliser tous les scripts et contenant

1. les bases de données (fichiers .sql)
2. les fichiers shape ou raster nécessaires à la réalisation des cartes de la partie II.

Ce cours et ses annexes sont gratuits et distribués sous une [Licence Creative Commons Attribution-ShareAlike 3.0 Unported](#). Ils ne viennent avec aucune garantie de quelque sorte et ni moi-même, ni mon université, ni mon institut de recherche ne saurions être tenus responsables des effets indésirables de toute mauvaise utilisation ou mauvaise compréhension de ces notes et de leur contenu. Je serai heureux de recevoir vos commentaires et suggestions.

Note de la troisième édition Cette édition présente quelques nouveautés qui concernent la programmation en python :

- les connections par le biais de la librairie `sqlalchemy` et son usage avec `pandas` sont plus fréquentes ;
- Les scripts cartographiques sont dorénavant réalisés avec `cartopy`. `cartopy` a maintenant remplacé `basemap` comme librairie de référence en cartographie. Pour une utilisation de `basemap` on pourra se référer à une ancienne édition de ce cours ;
- le chapitre sur les cartes web avec la librairie `folium` s'est enrichi de quelques cartes.

Première partie
Bases de données

Chapitre 1

Introduction

1.1 Questions (réponses)

Transcription d'une première discussion avec l'assistance :

1. Qui sait ce qu'est une base de données relationnelle ? (Pas de réponse)
2. Est-ce que les noms d'Oracle, MySQL, SQLite, SQL Server, Access vous parlent ? (Oracle oui vaguement)
3. Qui a déjà utilisé une base de données relationnelle ? (Personne ou presque)
4. Qui a un compte Facebook ou un compte bancaire ? (Tout le monde...)
5. Je reformule : Qui a déjà programmé une base de données relationnelle ? (Personne ou presque).

La plupart des serveurs internet commerciaux utilisent des bases de données relationnelles. Lorsque vous achetez des billets de train ou d'avion, louez une chambre d'hôtel, vendez sur eBay ou racontez vos derniers «secrets» au monde entier (à vos «amis») vous interagissez avec des bases de données. Eh bien, au minimum, il peut être utile pour vous de comprendre comment celles-ci fonctionnent afin que vous soyez moins naïfs lorsque votre dernière image «privée» finit très bien classée dans une recherche Google, ou quand vous commencez à réaliser que la publicité pour des chaussettes rouges apparaît toujours depuis que vous avez évoqué cette paire que vous portez pour être chic dans un blog.

Maintenant quelques questions afin de poursuivre la réflexion

1. Lorsque vous acquérez des données, qu'en faites-vous ?
2. Pourquoi ?
3. Qu'est-ce qui est bien avec Excel ou Matlab ?
4. Quel est l'élément unitaire dans Excel ?
5. N'est-ce pas un problème ?

Ce qui finit toujours par arriver : vous finissez par mélanger des colonnes de données chèrement acquises avec des colonnes de calculs, des graphiques et des commentaires. Tout se mélange. Le risque augmente alors que vous perdez la connaissance de ce qui est et de ce qui n'est pas une donnée originale.

Si vos données ont une valeur quelconque, elles doivent être stockées quelque part (sur un ordinateur, un serveur) et protégées contre tout type de corruption ou de perte.

1.2 Objectifs

1. Savoir créer, gérer et utiliser des bases de données SQL (en utilisant MySQL ou MariaDB).
2. Savoir extraire efficacement des informations à l'aide de requêtes.
3. Savoir le faire depuis un langage de programmation (en utilisant Python).
4. Savoir tracer des cartes de qualité (en utilisant Python et MySQL)

Chacun de ces points pourrait constituer un cours. Le but ici est de montrer l'intérêt pour vous, futurs scientifiques, d'utiliser des bases de données dès que les données que vous utilisez pour acquérir sont autre chose qu'une simple table de quelques dizaines de valeurs.

J'ai choisi deux langages de programmation

- SQL est le langage standard de programmation de bases de données.
- Python est devenu en quelques années un langage de référence, notamment en sciences de la terre et en géomatique. Tous les grands logiciels de cartographie proposent des interfaces python.

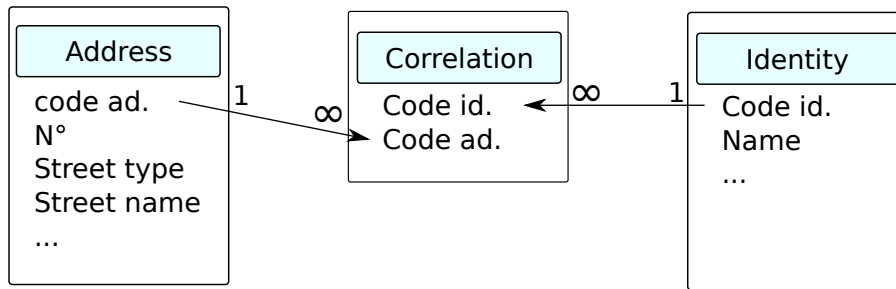


FIGURE 1.1 – Schéma d'un carnet d'adresses

- SQL (MySQL ou MariaDB) et Python sont indépendants des systèmes de vos ordinateurs
- MariaDB et Python sont libres!

On pourrait ajouter que MySQL¹, et Python sont très puissants et reconnus comme tels. Ils sont relativement simples dans le sens où, je l'espère, vous devriez être indépendants à la fin du cours et pouvoir développer vos propres applications. Ils n'évitent pas les difficultés liées à l'utilisation des bases de données et des langages orientés objet et c'est bien ainsi. Des dizaines de milliers de sites Web ont été développés avec MySQL et Python est probablement devenu le langage de programmation le plus polyvalent. (science, jeux, web etc ...). La communauté des utilisateurs est très importante et gère de nombreux forums d'information et de discussion.

1.3 Ma première base de données

Commençons par une base de données très simple (du moins supposée l'être) : un carnet d'adresses. Une adresse est une suite d'informations permettant au facteur de trouver un endroit quelque part sur la terre.

Il y a plusieurs façons d'écrire une adresse. Certaines sont très simples d'autres plus complexes. On peut par exemple décrire une adresse avec un pays, un code postal, une ville et certaines autres informations. L'important pour nous est que dans une table d'adresses, normalement, il n'y a pas de place pour une personne. En effet une adresse peut correspondre à plusieurs contacts et un contact peut avoir plusieurs adresses (domicile, bureau, vacances etc ...).

Vous avez donc besoin d'une table pour stocker les adresses et d'une table pour les contacts où vous stockez les noms de personnes. Mais alors vous devez être en mesure de relier les deux afin de dire qui vit ou travaille à quel endroit. Pour ce faire, vous devez identifier les personnes et les adresses de manière unique. Il y a plusieurs façons de le faire, mais la plus simple consiste à attribuer un code numérique qui caractérise de manière unique chacun de vos contacts ou adresses. C'est ce qu'on appelle une clé primaire (pensez aux index de Pandas). Une fois que vous avez fait cela, vous devez créer une table de corrélation où le numéro de contact 1 peut être atteint aux adresses numéro 2 et 10, par exemple. La structure d'une telle base de données ressemble à celle de la Fig. 1.1. Bien que cela puisse paraître un peu complexe pour un simple carnet d'adresses, cette structure est très efficace, en ce qu'elle permet de limiter les erreurs de saisie et les redondances. Elle permet en outre de faire bien d'autres choses encore. Dans les chapitres suivants, nous commencerons par des bases de données très simples avec une seule table, puis nous plus complexifierons à mesure que nos compétences augmenteront.

Une base de données pour nous se résumera à un ensemble de tables liées entre elles par des liens logiques matérialisés par des champs communs.

1. MySQL n'est plus opensource. Pour un équivalent opensource, vous pouvez installer MariaDB qui utilise la même syntaxe.

Chapitre 2

Créer et utiliser une base

2.1 Objectifs

- prise en main de MySQL et du langage SQL,
- Création de base et de table,
- Gestion des données

Les exemples choisis sont volontairement très simples.

2.2 Créer la base Toto

La première chose à faire est de se connecter au serveur de bases de données

```
# mysql -u root -p
# password :
mysql>
```

Si le serveur n'est pas sur votre machine il faudra rajouter l'adresse précédé de '-h'. Une fois connecté, pour savoir combien de bases sont enregistrées sur le serveur on écrira

```
1 show databases;
```

NB : toute instruction SQL se finit par un «;»

Pour se connecter à une base (test par exemple) on écrira

```
1 use test;
```

Pour créer la base Toto puis l'utiliser nous écrirons

```
1 create database Toto;
2 use Toto;
```

Pour détruire une base on écrira enfin

```
1 drop database Toto;
```

2.3 Créer la table tata

Une fois dans la base on peut obtenir la liste des tables que celle-ci contient

```
1 show tables;
```

Ici il n'y en a pas. Nous allons donc en créer une que nous appellerons **tata**. Contrairement à une base, lors de la création d'une table nous devons définir la table de façon détaillée. Nous devons ainsi dire de quoi (quels champs) seront composés les enregistrements. Dans notre cas chaque enregistrement sera composé de trois champs : val1 (un nombre entier), val2 (un nombre entier) et val3 (un nombre réel).

```
1 create table tata (val1 int, val2 int, val3 float);
```

pour connaître la structure de **tata** on écrira

```
1 describe my_table;
```


Attention à la casse pour les noms de base et de table. `Tata` n'est pas la même chose que `tata`. `tata` est composée de deux types de données des entiers `int` et de réels `float`. Il en existe d'autres. Les plus communs sont

- Les valeurs numériques :
 - BIT ou BOOL pour Boolean (yes/no true/false). Correspond à des entiers de 1 byte;
 - DOUBLE pour des réels double précision;
 - On peut ajouter UNSIGNED pour spécifier que les valeurs sont positives;
- Le temps :
 - DATE date (sans les heures);
 - TIME temps (HMS.xxx sans les dates);
 - DATETIME les deux!
- Les chaînes de caractère :
 - CHAR(N) : chaîne de taille fixe ($0 < N < 255$);
 - VARCHAR(N) : chaîne dynamique de taille ($0 < N < 255$);
 - BLOB ou TEXT, MEDIUMBLOB, LONGBLOB champs de 2^{16} , 2^{24} or $2^{32} = 4Go$ caractères respectivement.

Enfin supprimer une table se fait au moyen de la commande

```
1 drop table tata;
```

2.4 Modifier la structure d'une table

On modifie la structure d'une table au moyen d'une commande ALTER TABLE.

Pour changer le nom d'un champs :

```
1 alter table tata change val1 valeur1 int;
```

notez que l'on est obligé de redéfinir les caractéristiques du champs

Pour modifier la définition d'un champs sans changer son nom

```
1 alter table tata modify val2 int not null;
```

le champs `val2` ne pourra pas être laissé vide lors de la saisie des enregistrements.

Pour ajouter une colonne nommée `nom` et de te type `varchar` :

```
1 alter table tata add column nom varchar(55);
```

Enfin pour ajouter une colonne d'indexation automatique :

```
1 alter table totofill add column code int auto_increment not null primary key;
```

Pour effacer une colonne

```
1 alter table tata drop column nom;
```

2.5 Enregistrer des données

C'est le but d'une base de données : enregistrer et conserver vos données. Ajoutons 4 enregistrements contenant chacun trois valeurs, deux entières et une réelle

```
1 insert into tata values (0,1,1.0) ,(1,2,3.1) ,(2,3,4.3) ,(3,4,7.5);
```

On voit vite que cela peut devenir fastidieux. Afin d'éviter ce problème de saisie on peut charger un fichier texte contenant les données sous une forme structurée (comme un fichier.csv par exemple). On écrira ainsi

```
1 /* if separator is a tab (default) */
2 load data local infile "nom_du_fichier" into table tata
3 /* if separator is a comma */
4 load data local infile "nom_du_fichier" into table tata fields terminated by ',';
```

MySQL suppose alors que le fichier est dans le répertoire courant. Sinon il faut ajouter un chemin vers le fichier `"chemin_vers_tata/tata.txt"`. Si la fonction n'est pas acceptée cela veut dire qu'elle est par défaut désactivée par le serveur. Pour l'activer il faut vous déconnecter et vous reconnecter en écrivant

```
mysql -u votre_user -p --local-infile
```

Vous avez un fichier `tata.csv` contenant 100000 lignes de données. vous pouvez le charger avec

```
1 load data local infile "chemin/tata.csv" into table tata fields terminated by ',';
```

Enfin nous verrons comment accéder et insérer des données depuis un langage de programmation comme python plus tard.

2.6 Voir, effacer et mettre à jour

2.6.1 Voir

la commande `DESCRIBE` permet de voir la structure d'une table pas les données qui s'y trouvent. Pour voir vos enregistrements vous pouvez écrire

```
1 select * from tata;
```

Nous consacrerons tous les chapitres suivants à la commande `SELECT` car c'est le coeur du langage SQL et de loin celle que vous utiliserez le plus par la suite si vous travaillez avec de bases de données.

2.6.2 Effacer

Pour effacer des données, des colonnes, des tables ou des bases de données on utilise les commandes `DELETE` et `DROP`. Nous avons déjà vu la seconde pour les bases et les tables.

Il faut bien comprendre qu'on efface pas une donnée mais un enregistrement. Pour effacer un ou plusieurs enregistrements

```
1 delete from tata where condition
```

par exemple toutes les valeurs de tata dont val1=10 sont fausses, il faut effacer les enregistrements. On écrira

```
1 delete from tata where val1=10;
```

On verra que toutes les conditions applicables à la clause `where` de la commande `select` sont applicables à `delete`.

2.6.3 Mettre à jour

La mise à jour des données fonctionne sur le même principe que la sélection et l'effacement. Si par exemple les données val2 correspondant à une valeur val1 de 9 (on n'a effacé les données val1=10 rappelez vous...) doivent être modifiées on écrira

```
1 update tata set val2=val2+1 where val1=9
```

Question importante pour la suite : peut-on modifier les données d'une ligne particulière de tata ? Comment procède-t-on ? Que doit-on faire si on ne peut pas ?

2.7 Sauvegarder ou transférer une base

L'opération la plus simple consiste à créer un «dump» de la base dans un fichier texte. Pour cela, dans un terminal, entrez (sans aller à la ligne!)

```
1 mysqldump -u votre_nom_utilisateur> -p votre_mot_de_passe --databases
   nom_de_la_base_a_sauvegarder > nom_fichier.sql
```

un fichier `.sql` devrait apparaître dans le répertoire d'où vous avez lancé la commande. Si vous ouvrez ce fichier dans un éditeur de texte classique vous devriez voir apparaître quelque chose comme

```
1 -- MySQL dump 10.16  Distrib 10.1.26-MariaDB, for debian-linux-gnu (x86_64)
2 --
3 -- Host: localhost      Database: Parcelle
4 -- -----
5 -- Server version      10.1.26-MariaDB-0+deb9u1
6
7 /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
8 /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
9 /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
10 /*!40101 SET NAMES utf8mb4 */;
11 /*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
12 /*!40103 SET TIME_ZONE='+00:00' */;
13 /*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
14 /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
15 /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
16 /*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
17
18 --
```

```

19 -- Current Database: 'Parcelle'
20 --
21
22 CREATE DATABASE /*!32312 IF NOT EXISTS*/ 'Parcelle' /*!40100 DEFAULT CHARACTER SET
    latin1 */;
23
24 USE 'Parcelle';
25
26 --
27 -- Table structure for table 'imetos'
28 --
29
30 DROP TABLE IF EXISTS 'imetos';
31 /*!40101 SET @saved_cs_client      = @@character_set_client */;
32 /*!40101 SET character_set_client = utf8 */;
33 CREATE TABLE 'imetos' (
34   'date' datetime DEFAULT NULL,
35   'tsolav' float DEFAULT NULL,
36   'tsolmin' float DEFAULT NULL,
37   'tsolmax' float DEFAULT NULL,
38   'solrad' int(11) DEFAULT NULL,
39   'winddir' int(11) DEFAULT NULL,
40   'precip' float DEFAULT NULL,
41   'vwindav' float DEFAULT NULL,
42   'vwindmax' float DEFAULT NULL,
43   'leafwet' int(11) DEFAULT NULL,
44   'tairav' float DEFAULT NULL,
45   'tairmin' float DEFAULT NULL,
46   'tairmax' float DEFAULT NULL,
47   'relhum' int(11) DEFAULT NULL,
48   'dewav' float DEFAULT NULL,
49   'dewmin' float DEFAULT NULL,
50   'pairav' int(11) DEFAULT NULL,
51   'pairmin' int(11) DEFAULT NULL,
52   'pairmax' int(11) DEFAULT NULL
53 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
54 /*!40101 SET character_set_client = @saved_cs_client */;
55
56 --
57 -- Dumping data for table 'imetos'
58 --
59
60 LOCK TABLES 'imetos' WRITE;
61 /*!40000 ALTER TABLE 'imetos' DISABLE KEYS */;
62 INSERT INTO 'imetos' VALUES ('2007-12-14 04:00:00'
    ,16.1,15.7,16.4,0,222,0,0,0,0,17.08,17.05,17.13,34,0.9,0.6,1025,1025,1026), ('
    2007-12-14 05:00:00'
    ,15.8,15.6,16.4,0,222,0,0,0,0,17.01,16.93,17.07,34,0.5,0.5,1025,1025,1025), ('
    2007-12-14 06:00:00'
    ,15.7,15.2,16.3,0,222,0,0,0,0,16.9,16.85,16.96,33,0.5,0.5,1025,1025,1025), ('
    2007-12-14 07:00:00'
    ,15.6,15.4,15.9,0,222,0,0,0,0,16.74,16.67,16.83,33,0.4,0.3,1025,1025,1025), ('
    2007-12-14 08:00:00'
    ,15.5,15.3,15.8,0,222,0,0,0,0,16.63,16.58,16.7,33,0.1,-0.1,1025,1025,1025), ('
    2007-12-14 09:00:00'
    ,15.8,15.3,16.3,0,222,0,0,0,0,16.7,16.6,16.92,33,0.1,0,1025,1025,1025), ('
    2007-12-14 10:00:00'
    ,17.2,16.6,18.1,0,222,0,0,0,0,17.59,17.03,18.15,32,0.5,0.2,1025,1025,1025), ('
    2007-12-14 11:00:00'
    ,18.3,17.8,18.8,0,222,0,0,0,0,18.63,18.23,18.95,31,0.8,0.6,1025,1025,1025), ('
    2007-12-14 12:00:00'
    ,18.8,18.5,19.2,0,222,0,0,0,0,19.3,19.04,19.53,30,1.2,1.1,1025,1025,1025), ('
    2007-12-14 13:00:00'
    ,19,18.6,19.4,0,222,0,0,0,0,19.68,19.57,19.82,29,1,0.6,1024,1024,1024), ('
    2007-12-14 14:00:00'
    ,19,18.7,19.4,0,222,0,0,0,0,19.67,19.54,19.87,30,1.1,1,1023,1023,1024), ...

```

ces dumps sont très utiles à deux titres. Ils constituent une sauvegarde de votre base et permettent de la reconstituer sur n'importe quel ordinateur avec la commande suivante

```
1 > mysql -u votre_nom_utilisateur > -p votre_mot_de_passe < nom_fichier.sql
```


Chapitre 3

Les requêtes sur une table

3.1 La base de données Parcelle

Une station météorologique a été installée pendant une année à l'observatoire de Saint-Maur, près d'une parcelle expérimentale en plein air. Nous allons utiliser les données issues de cette station pour travailler sur notre première vraie base de données.

3.2 Structure et chargement de la table météorologique : imetos

La table unique de la base s'appelle `imetos`, c'est le nom de la station.

Les mesures sont toutes enregistrées en même temps. Toutes les valeurs de `date` sont donc uniques et caractérisent un unique instant de mesure. Ce champs, qui fonctionne un peu à la façon d'un index `Pandas`, est appelé une clé primaire de la table. Pour diverses raisons, par exemple une défaillance du capteur, nous devons laisser la possibilité aux autres champs de rester indéfinis ou `NULL`.

Afin de charger Parcelle, qui se trouve dans le fichier `Parcelle.sql`, sur votre serveur vous devez entrer la ligne suivante dans une fenêtre de commandes

```
1 mysql -u votre_user -p < chemin_vers/Parcelle.sql
```

le serveur vous demandera de saisir votre mot de passe et se chargera de lire le fichier `Parcelle.sql` et de créer une copie de la base.

si vous jetez un oeil au fichier `Parcelle.sql` vous retrouverez les commandes que nous avons découvertes au chapitre précédent. Evidemment la syntaxe est un peu plus complexe mais ne vous laissez pas impressionner ce n'est pas très compliqué.

3.3 Requêtes simples

3.3.1 Sélectionnez un ou plusieurs champs d'une table

Nous avons maintenant une base de données opérationnelle. Nous allons voir comment extraire des informations en répondant à des questions simples (ou complexes). On peut ainsi se demander combien d'enregistrements météorologiques il y a dans notre table? Quelle est la durée totale des précipitations? Quelle est la vitesse maximale du vent? quelles directions de vent ont-elles été enregistrées en janvier? Quelle est la moyenne des précipitations hebdomadaires sur mon site? etc etc etc ...

Pour cela, SQL a une commande magique : la commande `SELECT`, qui justifie à elle seule l'utilisation d'une base de données. Même pour une seule table!

Commençons par le début :

sélectionnez toutes les données disponibles dans `iMETOS`

```
1 select * from imetos;
```

Le caractère `*` signifie ici l'enregistrement dans son entier c'est à dire tous les champs de la table (ici un enregistrement est composé de 19 champs). Supposons maintenant que nous voulons seulement connaître les précipitations :

```
1 select precip from imetos;
```

L'extraction de plusieurs champs sera alors obtenue par



FIGURE 3.1 – L’observatoire de Saint-Maur en 2007

Nom du champs	Type de donnée & description	
date	datetime	date and time of measurement
tsolav	float	average soil temperature $^{\circ}C$
tsolmin	float	minimum over the measurement period
tsolmax	float	maximum over the measurement period
solrad	int	short wave solar radiation (W/m^2)
winddir	int	wind direction
precip	float	cumulative height registered by the auget(mm)
vwindav	float	average windspeed (m/s)
vwindmax	float	maximum windspeed
leafwet	int	leaf humidity (%)
tairav	float	average air temperature ($^{\circ}C$)
tairmin	float	minimum air temperature
tairmax	float	maximum air temperature
relhum	int	air relative humidity (%)
dewav	float	average dew point ($^{\circ}C$)
dewmin	float	minimum dew point
pairav	int	average air pressure ($mbar$)
pairmin	int	minimum air pressure
pairmax	int	maximum air pressure

TABLE 3.1 – Structure de la table imetos.

```
1 select field1, field2,... from mytable;
```

Ainsi la vitesse et la direction moyennes du vent sont obtenues par

```
1 select vwindav, winddir from imetos;
```

3.3.2 Ordonner

On peut ajouter un tri croissant à nos données de vent

```
1 select vwindav, winddir from imetos order by vwindav asc;
```

Il est possible de trier sur deux champs

```
1 select distinct vwindav, winddir from imetos order by vwindav asc, winddir desc;
```

L'argument `asc` est facultatif car le tri croissant est l'option par défaut du classement. Si vous souhaitez un tri décroissant vous devez l'indiquer avec `desc`.

3.3.3 Distinguer

Nous avons vu que nous avons beaucoup de valeurs `Null` de précipitations. Celles-ci sont mesurées par un pluviomètre à auget basculeur qui mesure les précipitations par incrément de 0.2 mm. Nous aimerions savoir quelles sont les différentes valeurs de précipitations enregistrées sur la parcelle sans voir s'afficher toutes ces valeurs redondantes. Dans ce cas nous écrivons

```
1 select distinct precip from imetos;
```

La clause `DISTINCT`, comme son nom l'indique, est utilisée pour ne conserver que des valeurs distinctes. si nous l'appliquons au vent, nous obtenons

```
1 select distinct vwindav from imetos;
```

Si nous appliquons maintenant cette clause pour distinguer la vitesse moyenne du vent, puis la vitesse moyenne du vent ET la direction, nous obtenons deux résultats significativement différents. En fait, `DISTINCT` s'applique à tous les attributs sélectionnés. Dans le premier cas, comme pour les précipitations, vous sélectionnez les valeurs distinctes de la vitesse moyenne du vent et vous découvrez que vous pouvez avoir des vents moyens supérieurs à 30 km/h. Dans le second cas par contre la requête

```
1 select distinct vwindav, winddir from imetos;
```

sélectionne des paires distinctes qui sont significativement plus nombreuses pour la même vitesse que le vent dans différentes directions.

3.3.4 Calculer

Vous n'êtes pas habitués à des vitesses en m/s? Qu'à cela ne tienne multipliez le champs `vwindav` par 3600 et divisez par 1000 soit un facteur multiplicateur de 3.6 pour obtenir la vitesse en km/h

```
1 select distinct vwindav*3.6 from imetos;
```

Nous verrons plus tard que les calculs effectués peuvent être bien plus complexes.

3.3.5 Exporter

Imaginez que vous êtes satisfait de votre requête et que vous souhaitez l'exporter pour travailler sur votre extraction sous votre logiciel de conception préféré! Ajoutez simplement

```
1 select vwindav, winddir from imetos into outfile 'file.txt';
```

Si vous le faites, MySQL cachera le fichier dans son répertoire source, dans ce cas, `/var/lib/mysql/Parcelle`. Et voici le problème : vous n'avez pas le droit de parcourir ce répertoire sans privilèges d'administrateur. Donc, dans la pratique, vous devez créer un dossier public dans votre répertoire personnel `/home` et lui accorder tous les droits (`chmod 777`) afin que mysql puisse y écrire des fichiers.

```
1 select vwindav, winddir from imetos into outfile '/home/metivier/public/vents.txt';
```


3.4 Les conditions

3.4.1 Comparaison

L'utilisation de la conditionnalité, à travers la clause WHERE dans la requête, ouvre un champ d'investigation infini de notre base de données météo. Supposons que je veuille savoir quelle est la vitesse du vent quand il pleut. Je veux sélectionner les valeurs du vent moyen à condition que la pluie soit supérieure à 0. On transcrit cela en SQL par

```
1 select vwindav from imetos where precip > 0;
```

Commentaires.

1. Par défaut, SQL suppose que `precip` provient de la même table que `vwindav` donc `imetos`.
2. Vous pouvez placer une condition sur un champ sans le sélectionner.
3. Vous devez comprendre que, pour exécuter la requête, le serveur effectue un test sur les précipitations. Si ce test est positif, la valeur correspondante de la vitesse du vent dans l'enregistrement est sélectionnée.
4. En SQL, les comparaisons se font au moyen des opérateurs suivants `=`, `<>`, `>`, `>=`, `<` et enfin `<=`.

Supposons maintenant que je veuille connaître le rayonnement le 22 janvier. Étant donné le format de la date que j'ai utilisé `'2008-01-22 01 :00 :00'`, je dois extraire de ce champ toutes les valeurs qui ressemblent à `'2008-01 -22 01 :00 :00 '`. Pour ce faire, SQL propose la clause `sc like` et j'écrirai

```
1 select date, solrad from imetos where date like '2008-01-22%';
```

Le caractère `%` est un caractère générique ou joker. il signifie que toute chaîne commençant le 22 janvier 2008 répondra aux critères. Si vous voulez un caractère générique unique, utilisez `_` comme dans l'exemple suivant

```
1 select date, solrad from imetos where date like '2008-01-22_1:00:00';
```

La clause `LIKE` peut être appliqué à n'importe quel champ. Mysql fera comme s'il s'agissait d'une chaîne. Soyez donc prudent.

Deux autres conditions sont souvent utilisées et pratiques : `IN` et `BETWEEN`. Dans le cas où nous étudions la seconde est simple. Il suffit de sélectionner les valeurs limites de la plage comme, par exemple, une humidité comprise entre 60 et 70 %

```
1 select * from imetos where relhum between 60 and 70;
```

Vous l'utiliserez si vous avez une liste de valeurs. Par exemple, si vous souhaitez extraire des valeurs d'humidité correspondant à des précipitations de 0,4 ou 1 mm, vous devez écrire

```
1 select precip, relhum from imetos where precip in (0.4,1);
```

Enfin il est possible et souvent pratique d'utiliser la négation `NOT`. Ainsi

```
1 select * from imetos where relhum not between 60 and 70;
```

renvoie les enregistrements correspondant à une humidité relative inférieure à 60% ou supérieure à 70%.

3.4.2 Conditions multiples : utilisation des conjonctions

Je souhaite extraire les valeurs du rayonnement reçu quand il n'y a pas de précipitations et quand l'humidité est inférieure à 70%. Je souhaite aussi savoir quand les mesures ont été prises. J'utiliserai pour ce faire une condition `WHERE` avec une conjonction `AND`

```
1 select date, solrad from imetos where precip = 0 and relhum <70;
```

Maintenant, si je veux connaître la même chose mais avec une condition moins stricte — par exemple souhaite connaître l'irradiation quand il n'y a pas de précipitations ou quand l'humidité est inférieure à 70%— j'utiliserai une conjonction `OR`

```
1 select date, solrad from imetos where precip = 0 or relhum <70;
```

Il y a enfin une troisième conjonction : le ou exclusif `XOR`. Cette conjonction est plus difficile d'emploi car

```
1 a xor b <=> (a and (not b)) or ((not a) and b).
```

Il n'y a pas de limite au nombre de conditions et donc de conjonctions. Faites cependant attention au sens de vos prédicats et recourez systématiquement à l'emploi de parenthèses (comme dans l'exemple précédent) pour vous assurer que le langage comprendra vos conditions comme vous l'entendez.

3.5 L'emploi de fonctions

Je vous donne une liste non exhaustive des fonctions les plus courantes de MySQL. Ces fonctions ne sont pas universelles. Chaque système de gestion implémente ses propres fonctions. Les fonctions peuvent apparaître dans les clauses SELECT, WHERE, GROUP BY, HAVING, ORDER BY (nous verrons celles que nous ne connaissons pas plus tard).

3.5.1 Fonctions concernant le temps

- YEAR(DATE), MONTH(DATE), WEEK(DATE), DAY(DATE) : retourne l'année, le mois, la semaine, le jour (du mois) d'un champs de type `date`.
- DAYOFYEAR(DATE), DAYOFWEEK(DATE) : jour de l'année ou de la semaine.
- HOUR(TIME), MINUTE(TIME) SECOND(TIME) : retourne les heures, minutes, ou secondes d'un champs `time` ou `datetime`.
- DATE_ADD(DATE, INTERVAL EXPR TYPE), DATE_SUB(DATE, INTERVAL EXPR TYPE) : ajoute ou soustrait un interval à un champs `date`

```
1 select date_add('2008-01-01', interval 1 day);
```

3.5.2 Fonctions de calculs

- ABS(X) : valeur absolue.
- COS(X), SIN(X), TAN(X), ACOS(X), ASIN(X), ATAN(X) : en radians...
- CEILING(X), FLOOR(X) : convertit en entier (arrondi à l'entier sup, ou inf).
- ROUND(X,D), TRUNCATE(X,D) : arrondi ou tronque à la décimale D. Par défaut arrondi (D=0)
- LN(X)=LOG(X), LOG2(X), LOG10(X) : logarithmes.
- EXP(X), SQRT(X), POW(X,Y) : exponentielle, racine et puissance.
- RADIANS(X), SIGN(X) : conversion de degrés vers radians et signe de x .

3.5.3 Fonctions concernant les chaînes de caractères

- ASCII(STR) : code ASCII du premier caractère de str.
- CHAR(N1,N2,N3,...) : fabrique une chaîne de caractère à partir de N1, N2, ...
- CHAR_LENGTH(STR) : longueur en octets de str.
- CONCAT(STR1,STR2,...) : concaténation, si une valeur est NULL retourne NULL.
- CONCAT_WS(SÉPARATEUR,STR1,STR2) : concaténation avec séparateur, ignore les NULL.
- LOCATE(SUBSTR,STR) : Position de la première occurrence de substr dans str. 0 si aucune.
- LTRIM(STR), RTRIM(STR) : tronque les espaces à gauche (resp. à droite).
- TRIM([BOTH | LEADING | TRAILING] [REMSTR] FROM] STR) : supprime le préfixe et/ou le suffixe remstr de str.
- SUBSTRING(STR,POS,LEN) : renvoie les LEN caractère à partir de la position POS de la chaîne str.

À titre d'exemple, nous voulons calculer la pression de vapeur saturante¹ (e_a^*)

$$e_a^* = 0.611 \exp\left(\frac{17.3T_a}{237.3 + T_a}\right) \quad (3.1)$$

La pression de vapeur (e_a) et la pression de vapeur saturante sont reliées par l'humidité de l'air (W_a) selon

$$\frac{e_a}{e_a^*} = W_a \quad (3.2)$$

On peut donc extraire la vapeur de pression en fonction de la date grâce à la requête suivante

```
1 select date, 0.01*relhum*0.611*exp(17.3*tairav/(237.3+tairav)) as ea from imetos;
```

Ce calcul peut bien entendu être effectué en python, comme on le verra par la suite.

3.6 L'agrégation de données

La possibilité d'agréger les données est une fonctionnalité essentielle de SQL. Elle permet d'effectuer des requêtes sur ce que l'on appelle les concepts latents d'une table. les mesures enregistrées dans `imetos` sont

1. Elle correspond à la pression de vapeur lorsque l'air est saturé et ne peut se charger en vapeur d'eau.

effectuées à des moments donnés enregistrés dans l'attribut `date`. Cet attribut contient diverses informations telles que l'année, le mois, le jour ou la semaine et peut faire l'objet d'une interrogation unique, par ex. Quelles sont les précipitations quotidiennes? Pour répondre à cela, nous devons combiner les mesures de précipitations effectuées chaque jour et les additionner. SQL peut le faire via une requête

```
1 select date(date), sum(precip) /* Extract date (YYYY-MM-DD)
2 from the date field and calculate $\sum_{0}^{23} p$ */
3 from imetos
4 group by date(date); /* sets the boundaries of the sum
5 calculated above */
```

Là encore, il n'est pas nécessaire de sélectionner le champ de date et la requête

```
1 select sum(precip) from imetos group by date(date);
```

fonctionne également et ne renvoie qu'un seul champ. Supposons que vous ne vouliez garder que des précipitations supérieures à 0, vous pouvez inclure une condition sur le résultat de SUM dans une clause HAVING

```
1 select date(date), sum(precip) from imetos group by date(date) having sum(precip) >
0;
```

Note importante : les conditions de sélections portant sur le résultat d'un calcul de regroupement doivent être regroupées dans une clause HAVING et NON dans une clause WHERE. Vous pouvez par contre ajouter d'autres conditions de sélections dans la clause WHERE et ordonner vos résultats comme vous le souhaitez.

Le regroupement peut être effectué sur plusieurs champs. Supposons que je veuille connaître la vitesse moyenne du vent pour chaque gamme de précipitations et pour chaque jour de mesure, je vais écrire

```
1 select date(date), precip, avg(vwindav) from imetos group by date(date), precip;
```

L'ordre dans lequel vous placez des champs dans la clause GROUP BY n'a aucune importance (devinez pourquoi?). Vous n'avez pas besoin d'utiliser une clause GROUP BY lorsque le calcul est effectué sur tous les enregistrements de la table, ainsi

```
1 select sum(precip) from imetos ;
```

sommera les précipitations de toute la table.

- AVG, STD : moyenne et écart-type,
- SUM, COUNT : somme et compte **ne pas confondre**,
- MIN, MAX...

La fonction COUNT est l'une des fonctions les plus populaires de la clause SELECT et elle est un peu particulière car elle peut être écrite de deux manières : COUNT (*) ou COUNT (FIELD). Dans le premier cas, le compte de tous les enregistrements non Null est effectué. Dans le second, seuls les enregistrements qui ont une valeur non Null du champ sélectionné sont comptés.

Ces deux requêtes renvoient le même résultat car il n'y a aucune valeur NULL dans les champs de la table `imetos`. La requête

```
1 select count(*), count(distinct precip), count(distinct relhum) from imetos;
```

Permet de comprendre la différence. Vous pouvez prendre le temps de réaliser que les possibilités offertes sont merveilleuses!

3.7 Applications

1. Précipitations du 5 janvier ?
2. Précipitations et vitesse du vent en cas de pluie ?
3. Différence de température entre le sol et l'air le 10 janvier ?
4. Combien d'enregistrements dans la table `imetos` ?
5. Combien de valeurs distinctes de pression (moyenne, min et max) ?
6. Combien de valeurs distinctes de précipitations ?
7. Valeur maximale des précipitations ?
8. Vitesse maximale du vent ?
9. Pression minimale ?
10. Pression (moyenne, min et max) en cas de pluie ?
11. Nombre d'enregistrements de précipitations non nuls pendant la journée ?

12. Nombre d'enregistrements de précipitations non nuls pendant la nuit ?
13. Pression moyenne lorsque l'air est saturé d'humidité ?
14. Quelles sont les valeurs moyennes, maximales et minimales de pression enregistrées ?
15. Quelle est la pression atmosphérique moyenne en cas de pluie ?
16. Quelle est l'humidité moyenne de l'air le 20 janvier ?
17. Quels sont l'écart moyen et standard des vents le 5 janvier ?
18. Quelles sont les précipitations hebdomadaires en janvier ?
19. Combien de jours la pluie a-t-elle duré plus de deux heures ?
20. Combien de valeurs distinctes de précipitations sont enregistrées chaque jour ?
21. Quels jours enregistrent plus de trois valeurs distinctes de précipitations ?
22. Quelle est la différence de température moyenne air-sol pendant la nuit ?
23. Quelle est la différence de température air-sol maximale pendant la journée ?
24. Quel est le gradient de température air-sol moyen entre le 15 et le 31 janvier ?
25. Combien de fois a-t-il gelé ?
26. Combien de jours a-t-il gelé ? (différent de la précédente)
27. Faire l'histogramme des directions du vent regroupées en classes de 20°.
28. Faites un histogramme de l'humidité dans les classes d'humidité de 10%.

3.8 Requêtes emboîtées

3.8.1 Principe

En raison de la structure d'une table, organisée en enregistrements dont les champs sont liés les uns aux autres, certaines requêtes doivent être effectuées par étapes successives. Par exemple, nous imaginons que nous voulons savoir quand le temps était le plus froid pendant la période de mesure. Pour cela, nous devons vérifier la valeur minimale de la température, puis rechercher le jour où cette température s'est produite. Si nous essayons de le faire en une seule étape selon

```
1 select date, min(tairmax) from imetos group by date;
```

cela ne marche pas. La raison provient de l'argument du GROUP BY. Un sous-ensemble est créé pour chaque «date», il y aura donc de nombreuses valeurs minimales. Pour répondre à la question, il faut d'abord trouver la valeur minimale de la table entière

```
1 select min(tairmax) from imetos;
```

puis utiliser la valeur retournée dans une condition WHERE :

```
1 select date, tairmin from imetos where tairmin in (select min(tairmin) from imetos);
```

Ce type de requête s'appelle une requête emboîtée et l'on comprend aisément pourquoi. Ces instructions (sélections incorporées entre parenthèses) peuvent être incluses dans les clauses FROM, WHERE et HAVING. Dans l'instruction FROM, la sous-requête doit être nommée à l'aide d'un alias. Dans l'exemple suivant, nous recherchons les dates lorsque la température est inférieure à 5 degrés de la valeur minimale :

```
1 select tairmin, tmin from imetos, (select min(tairmin) as tmin from imetos) as TM
   where tairmin between tmin and tmin+5
```

Les limites à l'emboîtement dépendent des logiciels utilisés. Les logiciels de cartographie (SIG), notamment, ont tendances à ne pas permettre les requêtes emboîtées. Afin de contourner ces limites on aura recours à des vues.

3.8.2 Applications

1. Quand le temps a-t-il été le plus chaud ?
2. Calculer la variation de température autour de la moyenne du jeu de données.
3. Normaliser les mesures de pression à la pression moyenne au 31 janvier.

3.9 Vues

Les vues sont des requêtes enregistrées dans la base de données lorsque vous les créez. Une fois créés, elles se comportent comme des tableaux virtuels. En effet seul le code qui permet leur création est effectivement enregistré. Elles peuvent être appelés dans d'autres requêtes comme s'il s'agissait de véritables tables. Elles permettent de «sauvegarder» les requêtes fréquemment utilisées, par ex. le calcul de la pression de vapeur saturantes pour ceux qui s'intéressent à l'évaporation. Dès qu'une base parle d'argent ou que les données ont une valeur autre que purement scientifique, les vues permettent de créer des sous-ensembles du jeu de données complet et de n'autoriser les utilisateurs qu'à consulter la vue et non l'ensemble complet des données. Imaginez ce qui se passerait si, quand vous quand vous vous connectez sur votre site bancaire, vous aviez accès à toutes les données bancaires des clients de votre banque...

La syntaxe de création est la suivante

```
1 create view tension(date,ess,esa) as
2 select date, 0.611*exp(17.3*tsolav/(237.3+tsolav)) as ess,
3 0.611*exp(17.3*tairav/(237.3+tairav)) as esa
4 from imetos;
```

Pour voir quelle table est une table ou une vue, vous devez saisir

```
1 show full tables;
```

ou

```
1 show full tables where table_type like 'view';
```

Afin d'établir une corrélation entre les champs d'une vue et ceux d'une table, il faut pouvoir déterminer quel enregistrement dans la vue correspond à ce qui est enregistré dans la table. Cette jointure (nous en avons parlé en introduction et nous y reviendrons plus en détail plus loin) peut être réalisé via un champ partagé. Ce champ doit également «identifier» les enregistrements des deux tables, c'est-à-dire être non-NULL (non vide) et avoir une valeur différente pour chaque enregistrement. Ici, le seul champ capable d'identifier un enregistrement de manière unique est `date`. Pour utiliser la vue écrite ci-dessus, il est essentiel d'inclure ce champs dans la vue.

Une fois cela fait, nous pouvons, par exemple, extraire les valeurs correspondantes de pression de vapeur saturante et d'humidité de l'air comme

```
1 select relhum, esa from imetos, tension where imetos.date = tension.date;
```

3.10 Conclusion

A la fin de ces deux premiers chapitres, vous devez savoir comment créer une base de données, créer une table dans cette base, la remplir avec des données et effectuer toutes sortes de requêtes plus ou moins simples. Si ce cours a atteint son objectif, vous devriez être convaincu, malgré l'interface peu conviviale avec laquelle vous travaillez, qu'une fois que vos données sont suffisamment nombreuses en quantité et en type, vous devrez laisser tomber votre tableur préféré.

Chapitre 4

Tables multiples, clés et liens

4.1 La base Parcours

Nous allons maintenant créer une base de données pour répondre à une question ou remplir une tâche. Nous souhaitons ainsi créer et organiser une base de données simple destinée à suivre le cursus des étudiants depuis leur baccalauréat et avant leur arrivée au département des sciences de la Terre.

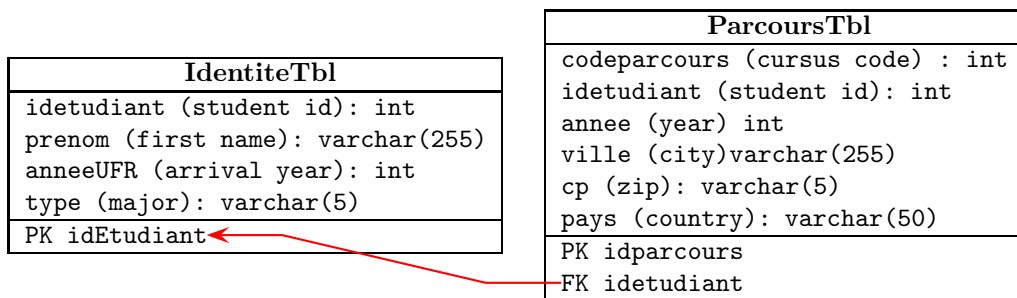


FIGURE 4.1 – Schéma de la base Parcours

4.1.1 Structure de la base

Pour créer la base, analysons notre problème. La question porte sur deux sujets.

1. les étudiants
2. les cursus (ou parcours)

Ces deux sujets ont leurs propres propriétés (qui définissent, caractérisent, un étudiant ou une année de cursus) mais ils sont liés. Chaque étudiant a son propre cursus, il est probablement unique mais pas toujours, il peut être assez complexe et s'étendre sur plusieurs années, il est donc important de définir une limite à ce que nous rechercherons. Les années correspondant au Bac et à l'arrivée au département constituent des limites naturelles.

Deuxièmement, nous devons caractériser nos «sujets». Nous ne serons pas exhaustifs, mais nous nous limiterons à ce qui est nécessaire pratiquer le SQL tout en répondant à des questions intéressantes. Pour les **étudiants**, des informations utiles peuvent être les suivantes :

- la formation choisie (génie de l'environnement, géophysique, télédétection ou autre) pouvant être utile pour essayer de comprendre s'il existe des différences significatives dans le cursus en fonction de la spécialisation ;
- l'année d'arrivée au département (de première arrivée en fait) ;
- le prénom (nous n'entrons pas dans le détail des noms de famille afin de conserver le caractère anonyme de la base) ;
- nous devons pouvoir distinguer un élève de l'autre, car chaque élève est par essence «unique» et possède un cursus lui aussi «unique». Un moyen simple de le faire est d'accorder à l'étudiant un code personnel (au même titre que le N° de sécurité sociale ou le N° d'étudiant).

Nous procédons ensuite pour le `parcours`. Compte tenu de la mission, nous avons besoin d'une année et d'un lieu. L'année la plus ancienne sera la dernière année du lycée lorsque le bac sera passé. Par conséquent on utilisera les attributs suivants :

- année,
- ville,
- code postal,
- pays,
- un identifiant permettant de distinguer les enregistrements.

Mais ce n'est pas assez. Nous devons relier les tables. Par conséquent, matérialiser le fait que chaque élève «possède» certains des enregistrements du cursus. Un seul champ peut le faire avec le numéro d'identification de l'étudiant. *In fine*, la structure de la base de données ressemble à la figure 4.1. On notera que le champ `idetudiant`, qui est unique et caractérise l'élève dans la table des étudiants, s'appelle une clé primaire. Il en va de même pour `codeparcours` dans la table `ParcoursTbl`. On notera en outre que `idetudiant` n'est plus unique dans la table `ParcoursTbl` car un étudiant peut avoir un cursus formé de plusieurs années. On appelle alors ce champs une clé étrangère, rappelant ainsi qu'elle pointe vers un champ qui est une clé primaire dans une autre table de la base de données.

La relation entre les deux tables est dite de un à plusieurs car un étudiant peut avoir plusieurs enregistrements dans `ParcoursTbl`.

Cette base de données est volontairement simple et non normalisée. La normalisation de la base de données est la procédure par laquelle les tables et leurs champs respectifs sont organisés afin de réduire au maximum la redondance des informations. Ce processus d'atomisation minimise les erreurs mais introduit de la complexité dans la base de données en augmentant le nombre de tables et de relations entre elles. Dans notre cas à chaque fois qu'un étudiant sera passé par «Paris», il faudra saisir cette information dans le champs ville de l'enregistrement correspondant, si l'étudiant se trompe et écrit «Pari», l'erreur se répercutera dans les résultats de nos requêtes. La normalisation vise à aussi à limiter ces risques. Selon la taille et les objectifs de votre base de données, elle peut être nécessaire. La figure 4.2 montre un exemple d'amélioration possible du schéma de la base. Cependant ce, schéma étant moins pratique pour nos besoins nous conserverons le schéma initial.

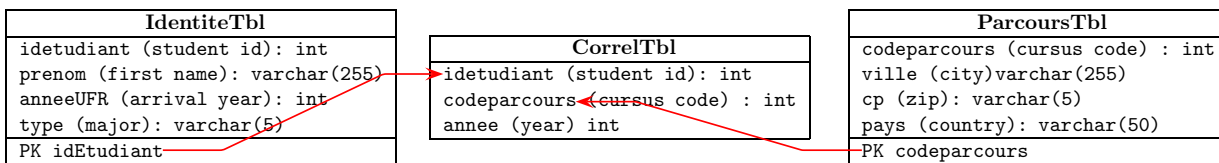


FIGURE 4.2 – Parcours : Schéma alternatif

4.1.2 Implémentation

Une fois la structure établie, l'implémentation est assez simple et se compose d'une série de commandes SQL que nous avons déjà vues.

```

1  /* create database */
2  create database Parcours;
3
4  /* create table Identitetbl */
5  create table Identitetbl(
6  idetudiant int auto_increment primary key,
7  prenom varchar(255),
8  anneeUFR int,
9  type varchar(3)
10 );
11
12 /* create table Parcours */
13 create table ParcoursTbl(
14 codeparcours int auto_increment primary key,
15 idetudiant int not null,
16 annee int not null,
17 ville varchar(255),
18 cp varchar(5),

```

```
19 Pays varchar(255)
20 );
```

4.1.3 Insérer les données

```
d'
1
2 /* once the session is started select the database and fill it */
3 /* fill a student ID first */
4 insert into IdentiteTbl values ('', 'françois', 1992, 'GEOPH');
5
6 /* Get the id of student*/
7 select Last_insert_id();
8
9 insert into ParcoursTbl values ('', value given by preceding query,
10 'ville', 'cp', 'pays');
11 ...
```

Si vous avez commis des erreurs et que vous souhaitez vous corriger en modifiant certaines valeurs ou en supprimant un enregistrement dans le fichier `ParcoursTbl` par exemple, l'une des méthodes les plus sûres est de trouver ces erreurs en utilisant quelques instructions `SELECT`. Le but de ces instructions sera de vous donner les valeurs des clés primaires dont vous avez besoin pour mettre à jour ou supprimer un (et un seul) enregistrement (ou plusieurs, mais les bons s.v.p). Une fois que les valeurs de clés sont connues, vous pouvez commencer la mise à jour ou la suppression en utilisant les commandes `UPDATE` et `DELETE` que nous avons déjà vues.

1. pour mettre à jour un enregistrement :

```
2 update ParcoursTbl set field_to_modify_1=new_value,
3 field_to_modify_2=new_value...
4 where codeparcours=key_of_the_line_to_modify;
```

2. pour effacer un enregistrement :

```
5 delete from ParcoursTbl
6 where codeparcours=key_of_the_line_to_destroy;
```

Attention à ne pas vous tromper, il n'y a pas Ctrl-Z dans une base de données SQL.

4.2 Requêtes

4.2.1 Patrons de recherche avec REGEXP

Jusqu'à présent, nous n'avons vu deux caractères génériques (`%` et `_`) à utiliser pour l'écriture de modèles (ou patrons) dans les expressions utilisant la clause `LIKE`. Le premier remplace n'importe quel nombre de caractère (y compris aucun caractère), le second remplace un caractère quelconque. Ces deux «jokers» sont essentiels, mais nous pouvons utiliser des patrons plus subtils en utilisant la clause `REGEXP` (pour regular expression).

```
7 /* villes must contain p or a */
8
9 select * from ParcoursTbl where ville regexp '[pa]'
10
11 /* villes must contain pa */
12
13 select * from ParcoursTbl where ville regexp 'pa'
14
15 /* villes must contain d or e or f */
16
17 select * from ParcoursTbl where ville regexp '[def]'
18 select * from ParcoursTbl where ville regexp '[d-f]'
```

Si vous cherchez un nombre, utilisez par exemple `[0-9]`. Enfin, si vous voulez que l'expression soit recherchée au début ou à la fin, vous pouvez utiliser

```
19 /* villes must start by p, a or c */
20 select * from ParcoursTbl where ville regexp '^[p,a,c]'
21 /* villes must end by p, a or c */
22 select * from ParcoursTbl where ville regexp '[p,a,c]$'
```


Il existe bien d'autres possibilités que vous pouvez découvrir en consultant l'aide en ligne de MariaDB ou MySQL.

4.2.2 Requêtes sur plusieurs tables

La base de données est composée de deux tables avec un champ commun `idetudiant`. Il est donc naturel de répondre à des questions telles que : où étaient les étudiants un an avant de rejoindre le département des sciences de la Terre? Pour trouver la réponse, nous devons connaître l'année d'arrivée (dans `IdentiteTbl`) et trouver la ville où il a vécu l'année précédente (informations situées dans `ParcoursTbl`). Le lien entre les deux informations provient du fait que les enregistrements doivent avoir le même `idetudiant`.

La requête SQL la plus simple doit spécifier ce lien et ressembler à

```
23 select IdentiteTbl.idetudiant, prenom, ville, cp, pays
24 from IdentiteTbl, ParcoursTbl
25 where IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
26 and annee=anneeUFR-1;
```

Ici, le lien entre les deux tables est effectué via

```
27 where IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
```

Ce type de lien est appelé une jointure interne. Si, par exemple, un élève de `IdentiteTbl` n'a pas d'enregistrement dans `ParcoursTbl` ou si un étudiant de `ParcoursTbl` n'existe pas dans `IdentiteTbl`¹, alors les enregistrements correspondants sont appelés *enregistrements orphelins* et sont exclus de la requête.

Cette façon simple de faire une requête sur deux tables, si elle est toujours opérationnelle n'est plus canonique. La syntaxe en usage est plus explicite

```
28 select IdentiteTbl.idetudiant, prenom, ville, cp, pays
29 from IdentiteTbl inner join ParcoursTbl
30 on IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
31 where annee=anneeUFR-1;
```

ou bien encore

```
32 select IdentiteTbl.idetudiant, prenom, ville, cp, pays
33 from IdentiteTbl join ParcoursTbl
34 on IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
35 where annee=anneeUFR-1;
```

Pour inclure les enregistrements orphelins, vous devez utiliser des jointures dites externes. En fonction de l'enregistrement que vous souhaitez conserver la requête ressemblera à

```
36 select IdentiteTbl.idetudiant, prenom, ville, cp, pays
37 from IdentiteTbl right join ParcoursTbl
38 on IdentiteTbl.ide\chapter{LabelLines}
39 \label{sec:labellines}
40 \lstinputlisting[language=python]{../map/scripts/liblabels.py}
41 tudiant=ParcoursTbl.idetudiant
42 where annee=anneeUFR-1;
```

ou

```
43 select IdentiteTbl.idetudiant, prenom, ville, cp, pays
44 from IdentiteTbl left join ParcoursTbl
45 on IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
46 where annee=anneeUFR-1;
```

Devinez ce que signifient ces requêtes! Là encore, il ne devrait y avoir AUCUN enregistrement orphelin dans le cas de la base de données `Parcours`. Il y en a peut-être mais elles proviennent d'erreurs et doivent être corrigées. Ces requêtes peuvent ensuite être utilisées pour vérifier qu'aucun orphelin n'existe. Nous utiliserons presque exclusivement des jointures internes.

Enfin essayez la requête suivante

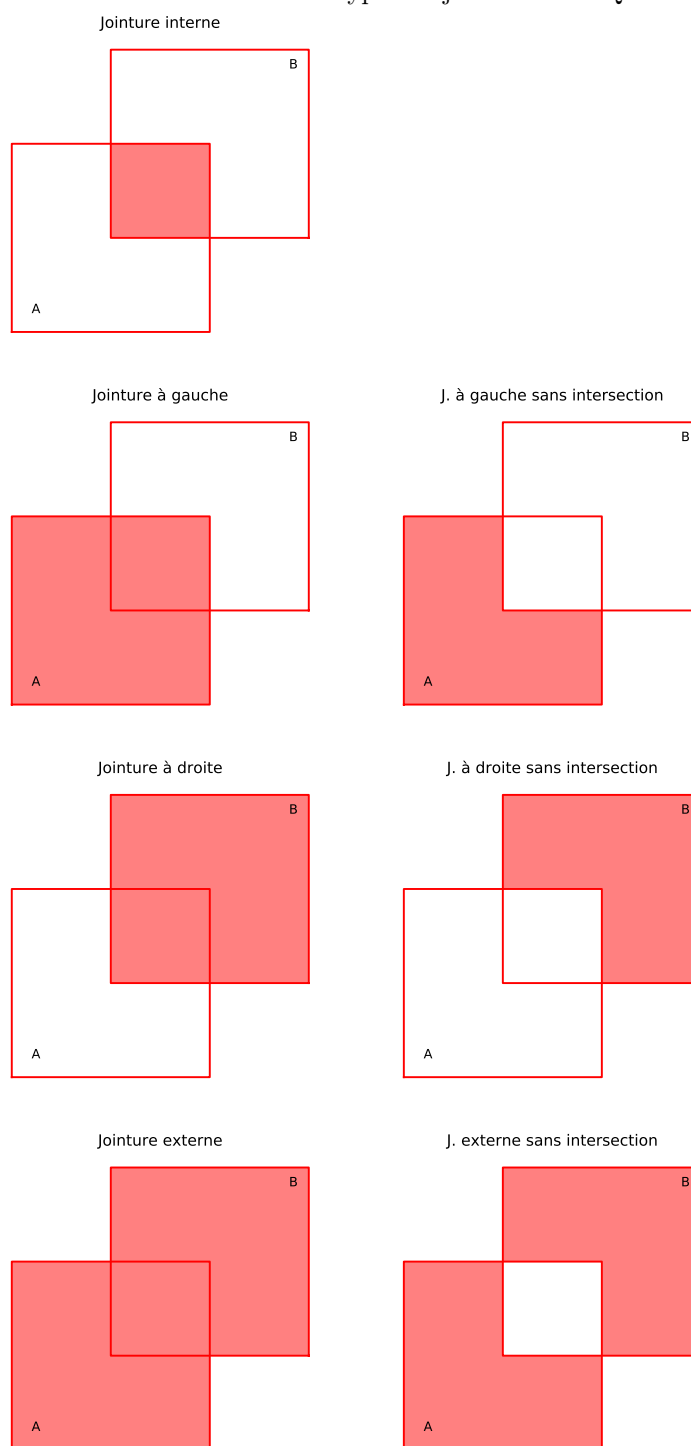
```
47 select IdentiteTbl.idetudiant, prenom, ville, cp, pays
48 from IdentiteTbl full join ParcoursTbl;
49 on IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
```

1. Cela ne devrait pas arriver, en particulier le cas des enregistrements dans `ParcoursTbl` qui ne correspondent pas à un élève inscrit dans `IdentiteTbl`.

4.2.3 Types de jointures

La figure 4.3 schématise les différents types de jointures entre ensembles (tables) que l'on peut faire en SQL. Les syntaxes de ces jointures sont données ci-dessous.

FIGURE 4.3 – Différents types de jointures en SQL



```
50 SELECT *
51 FROM A
52 INNER JOIN B ON A.key = B.key
```

```

53 SELECT *
54 FROM A
55 LEFT JOIN B ON A.key = B.key

```

```

56 SELECT *
57 FROM A
58 LEFT JOIN B ON A.key = B.key
59 WHERE B.key IS NULL

```

```

60 SELECT *
61 FROM A
62 RIGHT JOIN B ON A.key = B.key

```

```

63 SELECT *
64 FROM A
65 RIGHT JOIN B ON A.key = B.key
66 WHERE B.key IS NULL

```

```

67 SELECT *
68 FROM A
69 FULL JOIN B ON A.key = B.key

```

La jointure externe complète (Full outer join) n'est pas implémentée en MySQL. Pour vous en sortir vous devez procéder comme suit :

```

70 (SELECT * FROM A LEFT JOIN B ON A.key = B.key)
71 UNION
72 (SELECT * FROM A RIGHT JOIN B ON A.key = B.key)

```

Au passage vous aurez compris que vous avez la possibilité de regrouper les résultats de deux requêtes...

```

73 SELECT *
74 FROM A
75 FULL JOIN B ON A.key = B.key
76 WHERE A.key IS NULL
77 OR B.key IS NULL

```

Idem elles ne sont pas implémentées dans MySQL. Vous pouvez vous en sortir comme indiqué précédemment.

```

78 SELECT *
79 FROM A
80 CROSS JOIN B

```

4.2.4 Requêtes simples sur une ou deux tables

Produit croisé

1. Combien d'enregistrements y a-t-il dans la table IdentitéTbl ?
2. Combien d'enregistrements y a-t-il dans la table ParcoursTbl ?
3. Combien de Charlottes ?
4. Liste des étudiants dont le prénom commence par c ?
5. Quand sont arrivés les étudiants dont le prénom commence par b ou f ?

6. Combien d'étudiants ont leur prénom qui commence par d ?
7. Combien d'étudiants ont leur prénom qui comporte un h ?
8. Combien d'étudiants ont leur prénom qui finit par a ?
9. Combien d'étudiants sont arrivés à l'UFR
 - (a) En 2004
 - (b) Avant 2004
 - (c) Après 2003
 - (d) Avant 2002 et après 2004
10. Combien d'étudiants de GEI sont arrivés à l'UFR en 2004
11. Combien d'étudiants sont arrivés à l'UFR par année ?
12. Combien d'étudiants dans chaque cursus pour chaque année d'arrivée ?
13. Depuis combien d'années est arrivé le plus ancien étudiant de l'UFR ?
14. Quelles est la liste des étudiants qui ont au moins un homonyme ?
15. Ordonnez cette liste par type de cursus et par ordre alphabétique..
16. Quelle est la plus ancienne année enregistrée dans la table des Parcours ?
17. Combien y a-t-il d'étudiants dont le prénom commence par une lettre comprise entre d et g ?
18. Combien de prénoms commencent par une lettre comprise entre d et g ?
19. Combien d'étudiants de GEI sont arrivés à l'UFR en 2003
20. Combien ai-je d'enregistrements dans la table ParcoursTbl qui concerne des GEI ou des G2S ?
21. Où étaient les étudiants un an avant leur arrivée à l'UFR ?
22. Combien y avait-il alors d'étudiants par ville ?
23. Dans quelle ville étaient les étudiants de G2S 3 ans avant leur arrivée à l'UFR ?
24. Combien Y avait-il d'étudiant de GEI par ville 3 ans avant leur arrivée à l'UFR.
25. Dans quel département étaient les étudiants de G2S 2 ans avant leur arrivée à l'UFR ?
26. Combien d'étudiant de GEI sont arrivés à l'UFR en provenance du '91' ?
27. Combien d'étudiants sont arrivés à l'UFR en provenance du '92' ?
28. Classez les étudiants par ordre décroissant de mobilité.
29. Combien d'années se sont écoulées entre le bac et l'arrivée à l'UFR pour les étudiants de G2S ?
30. Quels sont les étudiants qui sont rentrés à l'UFR après leur bac et quand sont-ils arrivés ?
31. Liste des étudiants par type de parcours ayant résidé à Créteil ?
32. Liste des étudiants qui ont étudié à Paris et nombre d'années qu'ils y ont passé ?
33. Pour chaque étudiant combien d'années se sont écoulées entre le bac et leur arrivée
34. Pour chaque étudiant : depuis combien de temps a-t-il ou elle obtenu(e) son bac

4.2.5 Requêtes emboîtées

1. Combien d'années en moyenne les étudiants mettent-t-ils avant d'arriver à l'UFR ?
2. Combien de villes par départements ?
3. Qui a passé son bac avant tout le monde ?
4. Comment se prénomment les étudiants qui ont étudié à Paris avant d'arriver à l'UFR ?

Pour faire disparaître le second charlotte utiliser la clause DISTINCT.

Chapitre 5

Utiliser Python et MySQL

5.1 Introduction

La connexion à une base de données MySQL depuis Python se fait au moyen de bibliothèques dédiées. Nous allons en utiliser une appelée `MySQLdb.py`. Nous en utiliserons une autre `sqlalchemy` par la suite. La procédure de connexion est relativement simple et codifiée. Nous verrons les différentes étapes une à une. Tout se fait au moyen de classes (A.2.3), On crée une succession d'objets dont on utilise les fonctions par la suite.

La procédure est de façon quasi systématique la suivante

1. créer un objet connexion,
2. créer un objet facteur,
3. écrire son instruction SQL,
4. la poster par l'intermédiaire du facteur,
5. recevoir la réponse par l'intermédiaire du facteur,
6. traiter la réponse.

Une fois cela fait nous écrivons une classe simple qui permet de cacher les différentes étapes de connexion et d'effectuer l'essentiel des opérations dont nous avons besoin au quotidien.

Nota Bene : les scripts qui suivent sont prévus pour une connexion à une base de donnée locale (`localhost`) par l'utilisateur `root` muni du password `root`. Pensez à adapter ces données de connexion à votre situation pour que les scripts fonctionnent sans erreur.

5.2 Exemples

5.2.1 Récupérer des données numériques

Nous allons faire un histogramme du nombre d'inscrits en Génie de l'Environnement depuis la création de la base `Parcours`.

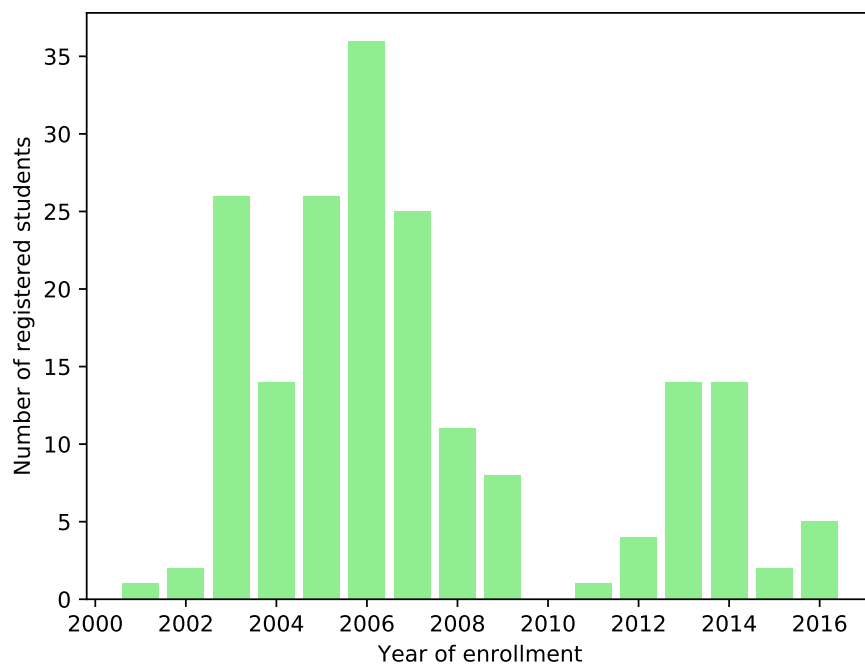
```
1 # -*- coding: utf-8 -*-
2 """
3 pm_1.py
4
5 First Python-MySQL script
6 """
7 import matplotlib.pyplot as plt
8 import MySQLdb as mysql
9
10 #get your connection identifiers
11 f=open('./identifiers.txt')
12 mylogin=f.readline().strip('\n')
13 mypass=f.readline().strip('\n')
14
15 # 1) Create connection
16 conn = mysql.connect(host="localhost", user=mylogin, db="Parcours", passwd=mypass)
17
18 # 2) Create postman (cursor)
19 postman = conn.cursor()
```

```

20
21 # 3) write SQL GEI
22 sql="select anneeUFR, count(*) from IdentiteTbl where anneeUFR>1900 and type='GEI'
      group by anneeUFR"
23
24 # 4) Send letter
25 postman.execute(sql)
26
27 # 5) Receive answer
28 res=py.array(postman.fetchall())
29
30 # 6) Process
31 anneeGEI=res.transpose()[0]
32 inscritsGEI=res.transpose()[1]
33
34 # graphique
35 fig=py.figure()
36 py.bar(anneeGEI, inscritsGEI, color = 'lightgreen')
37 py.ylabel(u"Number of registered students")
38 py.xlabel(u"Year of enrollment")
39 #py.savefig("../figures/FigSQL1.pdf", bbox_inches = 'tight')
40 py.show()
41
42 # close the connexion
43 postman.close()
44 conn.close()

```

FIGURE 5.1 – Nombre d'inscrits en Génie de l'Environnement depuis la création de la base Parcours



5.2.2 Récupérer des données mixtes

Nous souhaitons tracer les précipitations quotidiennes à l'observatoire de Saint-Maur en utilisant les données stockées dans la table `imetos` de la base de données `Parcelle`. Les scripts suivants se connectent à la base de données, récupèrent les données à l'aide d'une requête SQL et affichent les résultats.

```

1 """
2 pm_2.py
3
4 a simple Python script that
5 1) connects to the Parcelle database

```

```

6  2) queries the imetos table for daily precipitations
7  3) plots the retrieved data
8  """
9
10
11 # import statements
12 import MySQLdb # MySQL api
13 import matplotlib.pyplot as py
14
15
16 #get your connection identifiers
17 f=open('./identifiers.txt')
18 mylogin=f.readline().strip('\n')
19 mypass=f.readline().strip('\n')
20
21 # name of the database you wish to connect to
22 base = 'Parcelle'
23 # establish connection
24 conn = MySQLdb.connect(host="localhost", user=mylogin, passwd=mypass, db=base)
25 # create a cursor object to send queries
26 cursor = conn.cursor()
27 # write the SQL query
28 query = 'select date(date), sum(precip) from imetos group by date(date)'
29 # execute the query
30 cursor.execute(query)
31 # fetch the dataset
32 rows = cursor.fetchall()
33
34 # define the variables as an empty python list
35 da = []
36 pr = []
37 # for each record returned store the values
38 for row in rows:
39     da.append(row[0])
40     pr.append(row[1])
41
42 # plot the results
43 fig = py.figure(1)
44 py.bar(da,pr)
45 py.xlabel('Date')
46 py.ylabel('Precipitation in mm')
47
48 fig.autofmt_xdate() # command to get the dates plotted correctly
49 #py.savefig("../figures/pm_2.pdf", bbox_inches = 'tight')
50 py.show()
51
52 cursor.close()
53 conn.close()

```

Vous noterez dans le script une seconde façon de récupérer les données retournées par le facteur. L'utilisation d'une écriture ou d'une autre dépend de la nature du jeu de données. Dans le cas de données numériques, la première est plus rapide. Elle ne peut par contre être utilisée dans le cas de données mixtes comme ici des nombres et des dates.

5.3 SQL DDL

Nous allons maintenant utiliser python pour créer une base de données, créer une table et y insérer des données. La procédure est quasiment la même que précédemment car toute instruction peut-être considérée comme une requête faite auprès du serveur qui l'accepte...ou pas. Le script suivant contient différentes fonctions, simples, qui permettent

1. d'établir une connexion à un serveur MySQL,
2. de créer une base,
3. de détruire une base,
4. d'exécuter une requête ne renvoyant pas de jeu de données (donc autre que `select`).

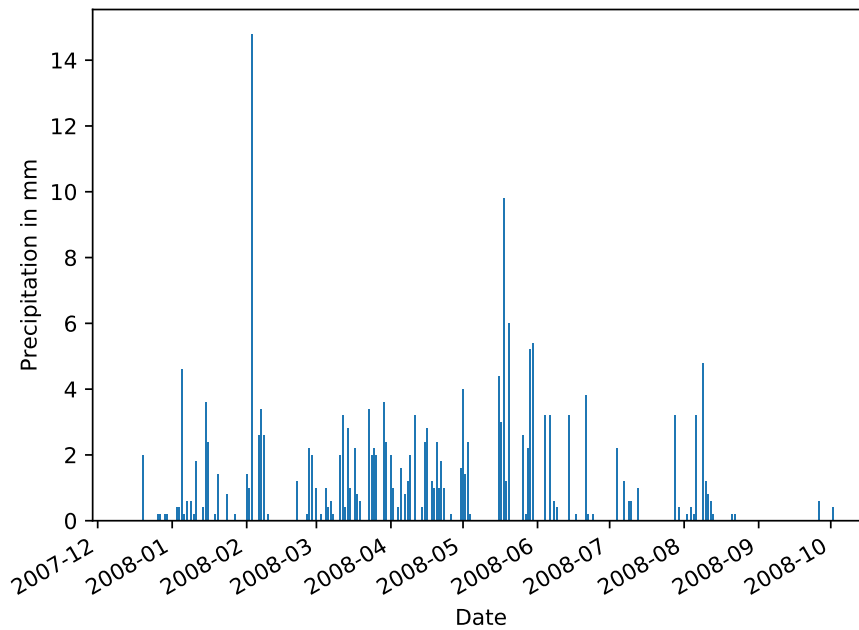


FIGURE 5.2 – Précipitations journalières à Saint-Maur des Fossés

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_3.py
4
5 Python-Mysql DDL
6 """
7 import MySQLdb
8
9 def create_connection(h = None, u = None, p = None):
10     """Creates and returns a connection object"""
11     Conn = None
12     if h is None or u is None or p is None:
13         print("You must valid host, user and passwords")
14     else:
15         try:
16             conn = MySQLdb.connect(host = h, user = u, passwd = p,
17                                   charset = 'utf8', use_unicode = True)
18         except:
19             print("connection failed")
20     return conn
21
22 def create_database(conn = None, dbname = None):
23     """
24     Establish connection to MySQL
25     Create database dbname
26     """
27     if dbname is None or conn is None:
28         print("You must provide a database name and valid connection")
29         return
30     else:
31         # create base
32         sql = "create database if not exists %s;" % dbname
33
34         cursor = conn.cursor()
35         res = cursor.execute(sql)
36         print(res)
37         conn.commit() # Needed in the case when you write something in the
38                       # database.
39         cursor.close()

```

```

38
39 def drop_database(conn = None, dbname = None):
40     """
41     Establish connection to MySQL
42     drop database dbname
43     """
44     if dbname is None or conn is None:
45         print("You must provide a database name and valid connection")
46         return
47     else:
48         # drop base
49         sql = "drop database %s;" % dbname
50         cursor = conn.cursor()
51         res = cursor.execute(sql)
52         print(res)
53         conn.commit() # Needed in the case when you write something in the database.
54         cursor.close()
55
56 def sql(conn = None, dbname = None, sqlstatement = None):
57     """
58     executes an sql statement (other than select) in database dbname
59     """
60     if dbname is None or conn is None or sqlstatement is None:
61         print("You must provide a database name and valid connection and sql statement")
62         return
63     else:
64         # create a cursor object to send queries
65         cursor = conn.cursor()
66         # connect to the database
67         cursor.execute("use %s;" % (dbname))
68         conn.commit()
69         # execute de query
70         cursor.execute(sqlstatement)
71         conn.commit()
72         cursor.close()
73
74     return conn

```

Ces fonctions sont très proches les unes des autres et peuvent être simplifiées (on le verra juste après) surtout si on a une connaissance du SQL. Voici quelques exemples d'utilisation de ces fonctions :

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_3-1.py
4
5 Python-Mysql DDL exemples using the library pm3
6 """
7
8 import pm_3
9
10 #get your connection identifiers
11 f=open('./identifiers.txt')
12 mylogin=f.readline().strip('\n')
13 mypass=f.readline().strip('\n')
14
15 #creates connection
16 myconn = pm_3.create_connection(h = "localhost", u = mylogin, p = mypass)
17
18 #creates database
19 pm_3.create_database(myconn, "Toto")
20
21 #
22 #sql queries
23 #
24
25 #create table

```

```

26 pm_3.sql(conn = myconn, dbname = "Toto", sqlstatement = "create_table_if_not_exists_
    tata(val1_int, val2_int, val3_float)")
27
28 #insert data
29 query = "insert_into_tata_values_(1,2,3.0),(2,3,4.0),(3,4,5.0)"
30
31 pm_3.sql(conn = myconn, dbname = "Toto", sqlstatement = query)
32
33 # close connection
34 myconn.close()

```

On voit que la réalisation des commandes `insert`, `update`, `delete` se fait de façon simple par l'envoi d'une requête au serveur.

5.3.1 Traiter les erreurs

Pour ceux qui connaissent un peu la programmation, il devient vite nécessaire d'essayer de gérer les erreurs. Le processus d'interrogation devrait donc être placé dans un bloc `try ... except` afin que le programme ne s'arrête pas en cas d'échec de la requête. Le code ressemblera alors à

```

1 req="SELECT_*_from_mytable"
2 try:
3     mycursor.execute(req)
4 except Exception as err:
5     # send back the message
6     msg="Incorrect_SQL_Query_#####:\n%s\nError_detected:%"(req)
7     return msg+str(err)
8 res=[]

```

5.3.2 OpenDB : a simple wrapper

Un constat que l'on peut faire est que l'usage de `MySQLdb` conduit à l'ajout de lignes au script ce ne sont pas simples à lire et qui sont très répétitives. Il est donc préférable d'écrire une classe wrapper (enrobante).

La classe `OpenDB` permet de connecter exécuter une requête et de récupérer le résultat. Il peut également générer une requête dans un fichier texte si nécessaire.

```

1 #!/usr/local/bin/python
2 # coding: utf-8
3 # web, os and db
4 """
5 openDB.py
6
7 Class OpenDB: wrapper to connect and execute SQL queries (DDL and DML)
8 """
9 import operator, os, pickle, sys
10 import MySQLdb
11
12
13 class OpenDB:
14     """
15     class to encapsulate creation of connection and query execution to a Mysql
16     Database
17     """
18     def __init__(self, base = None, user = None, passwd = None, host = None):
19         if base is None or user is None or passwd is None or host is None:
20             print("you_must_provide_a_valid_database,,host,,user_and_password_to_
21                 create_an_OpenDB_instance")
22             return 1
23         else:
24             self.conn = MySQLdb.connect(host, user, passwd, base)
25
26     def execQuery(self, req, param = ()):
27         cursor = self.conn.cursor()
28         try:
29             cursor.execute(req, param)
30         except Exception as err:

```

```

29         # send back the message
30         msg = "Incorrect SQL Query: \n%s\nError detected: "%(req)
31         return msg+str(err)
32     res = []
33     if "SELECT" in req.upper():
34         res = cursor.fetchall()
35     else:
36         self.conn.commit()
37     cursor.close()
38     return res
39
40     def QueryToFile(self, req, fname = 'out.txt', param = ()):
41         """
42         Executes a query and outputs to a text file for further processing (if needed
43         )
44         """
45         res = self.execQuery(req, param)
46         f = open(fname, 'w')
47         for r in res:
48             for i in npy.arange(len(r)-1):
49                 st = "%s\t" % (r[i])
50                 f.write(st)
51             st = "%s\n" % (r[len(r)-1])
52             f.write(st)
53
54         f.close()
55
56     def close(self):
57         self.conn.close()

```

5.3.3 Précipitations : seconde implémentation

En utilisant la classe `OpenDB`, l'écriture des fonctions SQL se simplifie nettement et notre programme de récupération des précipitations devient alors

```

1  # -*- coding: utf-8 -*-
2  """
3  pm_4.py
4
5  same as pm_2.py but the data retrieval uses the OpenDB class
6  """
7
8  # import statements
9  import MySQLdb # MySQL api
10 from matplotlib.pyplot import *
11 from openDB import *
12
13
14 #get the data
15 #get your connection identifiers
16 f=open('./identifiers.txt')
17 mylogin=f.readline().strip('\n')
18 mypass=f.readline().strip('\n')
19
20 #establish the connection
21 o=OpenDB(base="Parcelle",host="localhost",user=mylogin,passwd=mypass)
22 #write and execute the query
23 sql="select date(date),sum(precip)from imetosgroupby date(date)"
24 rows=o.execQuery(sql)
25
26 # define the variables as an empty python list
27 da=[]
28 pr=[]
29 # for each record returned store the values
30 for row in rows:
31     da.append(row[0])

```

```

32     pr.append(row[1])
33
34 # plot the results
35 fig=figure(1)
36 bar(da,pr)
37 xlabel('Date')
38 ylabel('Precipitation_in_mm')
39
40 fig.autofmt_xdate() # command to get the dates plotted correctly
41 show()
42
43 #close database
44 o.close()

```

5.4 Applications

5.4.1 Cycles journaliers de température à Saint-Maur

Nous souhaitons tracer les températures journalières des cycles de température pendant les mois d'été (disons de juillet à septembre). Pour ce faire, nous devons extraire la moyenne horaire des températures de l'air. La requête correspondante est

```

1 select hour(instant), avg(tairav) from imetos
2 where month(instant) in(7,8,9)
3 group by hour(instant)

```

Le script n'est plus alors qu'une histoire de répétition

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_5.py
4
5 Plot of the hourly temperatures at Saint-Maur averaged over july, August and september
6 """
7 #import libraries
8 import matplotlib.pyplot as py
9 import openDB as odb
10
11 #get the data
12 #get your connection identifiers
13 f=open('./identifiers.txt')
14 mylogin=f.readline().strip('\n')
15 mypass=f.readline().strip('\n')
16
17 #establish the connection
18 o = odb.OpenDB(base = "Parcelle", user = mylogin, passwd = mypass, host = "localhost"
19 )
20 #write and execute the query
21 sql = "select hour(date), avg(tairav) from imetos where month(date) in (7,8,9) group
22 by hour(date)"
23
24 res = py.array(o.executeQuery(sql))
25 o.close()
26
27 #retrieve the results
28 h = res.transpose()[0]
29 ta = res.transpose()[1]
30
31 #plot the result
32 fig = py.figure()
33 py.plot(h, ta, 'ro-')
34 py.xlabel("Hour_of_the_day", fontsize = 14)
35 py.ylabel('Average_temperature_°C', fontsize = 14)
36 py.axis([0, 24, 10, 25 ])
37 py.grid(True)
38
39 py.show()

```

La figure 5.3 montre le résultat.

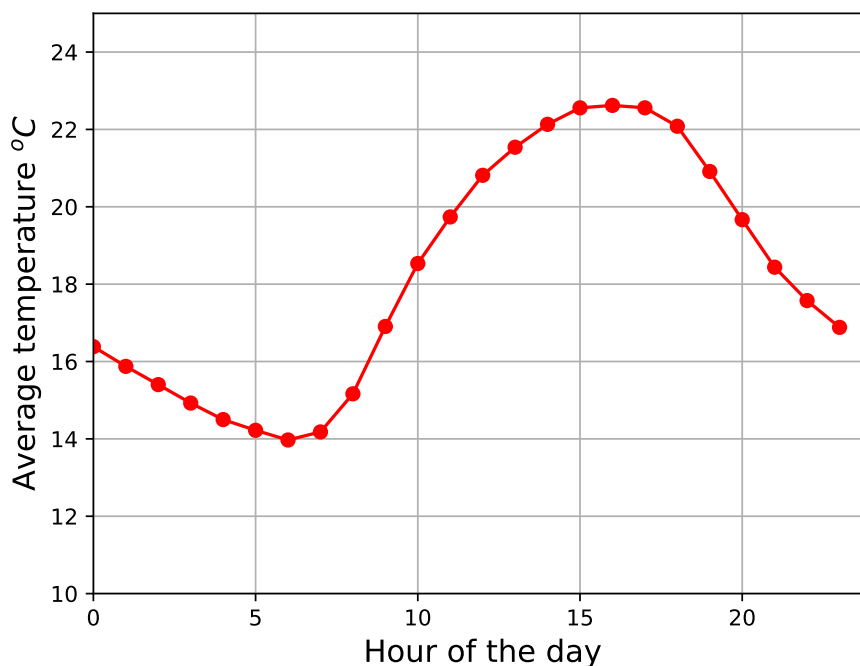


FIGURE 5.3 – Température journalière, durant l’été, à l’observatoire de Saint-Maur

5.4.2 Direction des vents at Saint Maur

Nous souhaitons établir et ordonner un histogramme des directions du vent regroupées en classes de 20° et le tracer. Une requête SQL permet de répondre à cette question. Il s’agit de

```
1 select round(winddir/20)*20, count(winddir) from imetos
2 group by round(winddir/20)*20
3 order by winddir;
```

Toute la difficulté vient du graphique en coordonnées polaires (figure 5.4). Et là n’oubliez jamais que `matplotlib` contient une aide extraordinaire sur internet.

```
1 # -*- coding: utf-8 -*-
2 """
3 pm_6.py
4
5 plots the histogram of wind directions at Saint-Maur as a polar plot.
6 """
7 #import libraries
8 import matplotlib.pyplot as py
9 import openDB as odb
10
11
12 #get the data
13 #get your connection identifiers
14 f=open('./identifiers.txt')
15 mylogin=f.readline().strip('\n')
16 mypass=f.readline().strip('\n')
17
18 #establish the connection
19 o = odb.OpenDB(base = "Parcelle", user = mylogin, passwd = mypass, host = "localhost"
20 )
21 #write and execute the query
22 sql = "select round(winddir/20)*20, count(winddir) from imetos group by round(winddir
23 /20) order by winddir"
24
25 res = o.execQuery(sql)
```

```

24 print(res)
25 #retrieve the results
26 radii = []
27 theta = []
28 for r in res:
29     if r[0] == 360: # add the 360 to the 0 class
30         radii[0] += r[1]
31     else:
32         theta.append(float(r[0]))
33         radii.append(r[1])
34
35 #plot the result
36 theta = py.array(theta)*py.pi/180.
37 radii = py.array(radii)
38
39 print(theta)
40
41 # force square figure and square axes as it looks better for polar plots
42 fig = py.figure(figsize=(8,8))
43 ax = fig.add_axes([0.1, 0.1, 0.8, 0.8], polar=True)
44
45 width = 20.*py.pi/180.
46 bars = py.bar(theta, radii, width = width, bottom = 0.0)
47 for r,bar in zip(radii, bars):
48     bar.set_facecolor("r")
49     bar.set_edgecolor("k")
50     bar.set_alpha(0.5)
51
52 py.show()

```

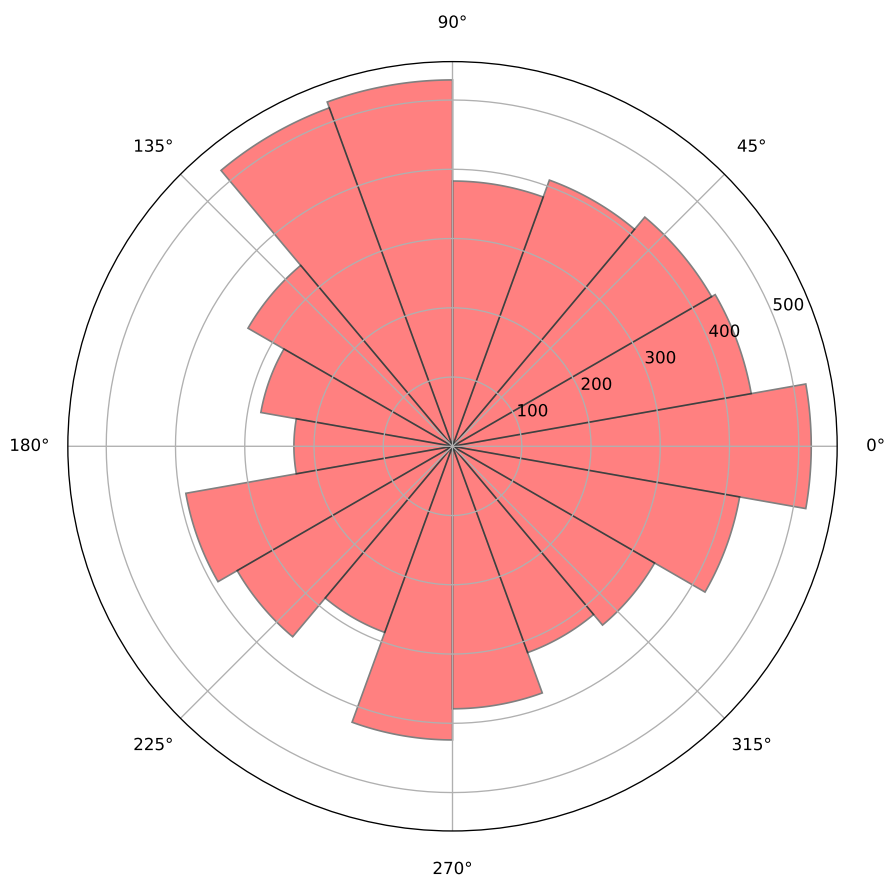


FIGURE 5.4 – Direction des vents à l'observatoire de Saint-Maur

Chapitre 6

Pandas et SQL

La librairie `Pandas` permet de faire énormément de choses et de se simplifier considérablement la vie quand on souhaite utiliser des données SQL. Elle constitue une alternative à l'utilisation de `MySQLdb`. Il est probable que c'est cette alternative que vous utiliserez le plus facilement. Pour bénéficier de ce chapitre vous devez déjà savoir utiliser `Pandas`. Si ce n'est pas le cas tous ce dont vous avez besoin pour démarrer se trouve sur le site suivant : <https://pandas.pydata.org/>

6.1 Connexion

6.1.1 SQLAlchemy

Afin d'utiliser `Pandas` pour traiter nos requêtes nous avons besoin d'une librairie de connexion adaptée. `Pandas` utilise `SQLAlchemy` qui est une alternative à `MySQLdb` que nous venons d'utiliser comme le montre le script suivant :

```
1 """
2 sqla_1.py
3
4 use sqlalchemy to connect to a Mysql database
5 """
6
7 from sqlalchemy import create_engine
8 mylogin="XXX"
9 mypass="YYY"
10
11 # default syntax engine = create_engine('mysql://user:passwd@host/dbname')
12 engine = create_engine('mysql://%s:%s@localhost/Parcelle' % (mylogin,mypass))
13 # establish connection
14 conn = engine.connect()
15
16 sql = "select_*_from_imetos_where_precip_>_0"
17 res = conn.execute(sql)
18
19
20 for row in res:
21     print (row)
```

Il est donc tout à fait possible de remplacer `MySQLdb` par `SQLAlchemy` dans les scripts du chapitre précédent (ce qui constitue un bon exercice).

6.1.2 Importer une table dans un dataframe

Pour se connecter à une base de données `Pandas` fait donc appel à `SQLAlchemy`. Le principe est très simple. On crée un objet de connexion que l'on passe à `Pandas` en argument. Après il se débrouille !

```
1 """
2 pm_7.py
3
4 use pandas to read an sql table from Mysql database
5 """
```



```

6 #library needed to create a connection that pandas can use
7 from sqlalchemy import create_engine
8 # pandas
9 import pandas
10
11 #get your connection identifiers
12 f=open('./identifiers.txt')
13 mylogin=f.readline().strip('\n')
14 mypass=f.readline().strip('\n')
15
16 # default syntax engine = create_engine('mysql://user:passwd@host/dbname')
17 engine = create_engine('mysql://%s:%s@localhost/Parcelle' % (mylogin, mypass))
18
19 df = pandas.read_sql_table('imetos', engine)
20 print(df.head())

```

6.1.3 Utiliser des requêtes

```

1 """
2 pm_7_2.py
3
4 use pandas to read sql data from Mysql database using a SELECT query
5 """
6 #library needed to create a connection that pandas can use
7 from sqlalchemy import create_engine
8 # pandas
9 import pandas
10
11 #get your connection identifiers
12 f=open('./identifiers.txt')
13 mylogin=f.readline().strip('\n')
14 mypass=f.readline().strip('\n')
15
16 # default syntax engine = create_engine('mysql://user:passwd@host/dbname')
17 engine = create_engine('mysql://%s:%s@localhost/Parcelle' % (mylogin, mypass))
18
19 sql="select year(date) as y, month(date) as m, sum(precip) as pcum from imetos group
      by month(date), year(date) order by year(date), month(date) "
20
21 df = pandas.read_sql(sql, engine)
22 print(df.head())

```

6.1.4 Charger des dataframes dans une base SQL

Il est aussi possible de stocker des tables `pandas` dans des bases SQL d'une façon très efficace. Le script suivant reprend notre table `tata` et y ajoute 100000 enregistrements pour les valeurs `val1`, `val2` et `val3`. Ces enregistrements sont créés à l'aide de variables aléatoires de distribution uniforme.

```

1 """
2 pm_9.py
3
4 use pandas to write pandas data to a MySQL table
5 you need to have the database Toto created see pm_3_1.py and pm_3.py
6 """
7 #library needed to create a connection that pandas can use
8 from sqlalchemy import create_engine
9 # pandas
10 import pandas
11 import numpy as np
12
13 #get your connection identifiers
14 f=open('./identifiers.txt')
15 mylogin=f.readline().strip('\n')
16 mypass=f.readline().strip('\n')
17
18 # connect to the Toto db

```

```

19 engine = create_engine('mysql://%s:%s@localhost/Toto' % (mylogin, mypass))
20
21 # create a dataframe with three series val1, val2 val3 of 100 000 values
22 # val 1 in [0,10]
23 val1 = np.random.randint(0,10,100000)
24 print(val1)
25 # val 2 in [0,100]
26 val2 = np.random.randint(0,100,100000)
27 print(val2)
28 # val 3 in [0,1000.0] uniform distribution
29 val3 = np.random.rand(100000)*1000.0
30 print(val3)
31
32 # create the frame
33 df = pandas.DataFrame({"val1": val1.tolist(), "val2": val2.tolist(), "val3": val3.
    tolist()})
34 print(df.head())
35
36 #store the data in table tata
37 df.to_sql('tata', engine, if_exists = 'append', index = False)

```

Vous pouvez vous connecter au serveur et vérifier que cela fonctionne puis, à partir des champs val1 et val2, vérifier que les distributions sont bien uniformes.

6.2 Applications

Nous reprenons ici les scripts développés précédemment avec MySQLdb avec la librairie Pandas. Le choix d'une méthode plutôt qu'une autre dépend essentiellement de vous. Si vous utilisez Pandas régulièrement et vous sentez à l'aise n'hésitez pas. Pandas offre des possibilités immenses et, qui plus est, présente des similitudes avec le SQL qui peuvent vous être utiles.

6.2.1 Précipitations : seconde implémentation

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_4p.py
4
5 same as pm_2.py and pm_4.py but the data retrieval uses SQLAlchemy and Pandas
6 instead of MySQLdb
7 """
8
9 # import statements
10 import matplotlib.pyplot as plt
11 from sqlalchemy import create_engine
12 import pandas
13
14 #get your connection identifiers
15 f=open('./identifiers.txt')
16 mylogin=f.readline().strip('\n')
17 mypass=f.readline().strip('\n')
18
19 # default syntax engine = create_engine('mysql://user:passwd@host/dbname')
20 engine = create_engine('mysql://%s:%s@localhost/Parcelle' % (mylogin, mypass))
21
22 sql = "select date(date) as d, sum(precip) as p from imetos group by date(date)"
23 df = pandas.read_sql(sql, engine)
24
25
26 # plot the results
27 fig=plt.figure(1)
28 plt.bar(df['d'].tolist(), df['p'].tolist())
29 plt.xlabel('Date')
30 plt.ylabel('Precipitation in mm')
31

```

```

32 fig.autofmt_xdate() # command to get the dates plotted correctly
33 py.show()
34
35 engine.close()

```

6.2.2 Cycles journaliers de température à Saint-Maur deuxième version

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_5p.py
4
5 same as pm_5.py but with SQLAlchemy and Pandas
6 """
7
8 #import libraries
9
10 import matplotlib.pyplot as py
11 from sqlalchemy import create_engine
12 import pandas
13
14 #get your connection identifiers
15 f=open('./identifiers.txt')
16 mylogin=f.readline().strip('\n')
17 mypass=f.readline().strip('\n')
18
19 # default syntax engine = create_engine('mysql://user:passwd@host/dbname')
20 engine = create_engine('mysql://%s:%s@localhost/Parcelle' % (mylogin,mypass))
21
22 #get the data
23 sql = "select hour(date) as h, avg(tairav) as ta from imetos where month(date) in
24       (7,8,9) group by hour(date)"
25 df = pandas.read_sql(sql, engine)
26
27
28 #plot the result
29 fig=py.figure()
30 py.plot(df['h'], df['ta'], 'ro-')
31 py.xlabel("Hour of the day", fontsize = 14)
32 py.ylabel('Average temperature °C$', fontsize = 14)
33 py.axis([0, 24, 10, 25 ])
34 py.grid(True)
35 py.show()
36
37 engine.close()

```

6.2.3 Direction des vents at Saint Maur

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_6.py
4
5 plots the histogram of wind directions at Saint-Maur as a polar plot.
6 """
7 #import libraries
8 import matplotlib.pyplot as py
9 import openDB as odb
10
11
12 #get the data
13 #get your connection identifiers
14 f=open('./identifiers.txt')
15 mylogin=f.readline().strip('\n')
16 mypass=f.readline().strip('\n')
17
18 #establish the connection

```

```

19 o = odb.OpenDB(base = "Parcelle", user = mylogin, passwd = mypass, host = "localhost"
20 )
21 #write and execute the query
22 sql = "select round(winndir/20)*20, count(winndir) from imetos group by round(winndir
23 /20)*20 order by winndir"
24
25 res = o.execQuery(sql)
26 print(res)
27 #retrieve the results
28 radii = []
29 theta = []
30 for r in res:
31     if r[0] == 360: # add the 360 to the 0 class
32         radii[0] += r[1]
33     else:
34         theta.append(float(r[0]))
35         radii.append(r[1])
36
37 #plot the result
38 theta = py.array(theta)*py.pi/180.
39 radii = py.array(radii)
40
41 print(theta)
42
43 # force square figure and square axes as it looks better for polar plots
44 fig = py.figure(figsize=(8,8))
45 ax = fig.add_axes([0.1, 0.1, 0.8, 0.8], polar=True)
46
47 width = 20.*py.pi/180.
48 bars = py.bar(theta, radii, width = width, bottom = 0.0)
49 for r,bar in zip(radii, bars):
50     bar.set_facecolor("r")
51     bar.set_edgecolor("k")
52     bar.set_alpha(0.5)
53
54 py.show()

```

6.3 les capacités de regroupement de Pandas

Qui plus est votre maîtrise des requêtes `select` doit vous permettre de maîtriser plus facilement les possibilités de `pandas`. La [documentation en ligne](#) de `pandas` présente une comparaison détaillée de la syntaxe de `pandas` et de la syntaxe SQL Le script suivant en reprend quelques une en les appliquant à `imetos`.

```

1 """
2 pm_8.py
3
4 use pandas to read sql data from Mysql database and compare syntaxes
5 """
6 #library needed to create a connection that pandas can use
7 from sqlalchemy import create_engine
8 # pandas
9 import pandas
10
11 #get your connection identifiers
12 f=open('./identifiers.txt')
13 mylogin=f.readline().strip('\n')
14 mypass=f.readline().strip('\n')
15
16 # default syntax engine = create_engine('mysql://user:passwd@host/dbname')
17 engine = create_engine('mysql://%s:%s@localhost/Parcelle' % (mylogin,mypass))
18
19 df = pandas.read_sql_table('imetos',engine)
20 print(df.head())
21
22
23 print("Number_of_hours_of_precipitation")

```

```
24 # select count(*) from imetos where precip > 0
25 print(df[df["precip"] > 0]["date"].count())
26
27
28 print("Number of hours of precipitation during night time")
29 # select count(*) from imetos where precip > 0 and solrad=0
30 print(df[(df["precip"] > 0) & (df["solrad"] == 0)]["date"].count())
31
32
33 print("Monthly Averaged hourly precipitation during daytime")
34 # select month(date), avg(precip) from imetos where solrad >0 group by month(date)
35 print(df[df["solrad"] > 0].groupby(df["date"].dt.month)["precip"].mean())
36
37 #etc...
```

Deuxième partie

Cartographie

Chapitre 7

Introduction

Une carte est constitué d'un fond (carte du monde, image satellite, carte routière, graticules, etc...,) auquel on ajoute des données brutes ou faisant l'objet d'une interprétation. Représenter cartographiquement des données veut dire être capable de traiter deux grandes familles de données : des données vectorielles (des points, vecteurs, lignes et autres polygones, souvent contenus dans ce que l'on appelle des shapefiles) et des données matricielles (appelées des rasters)¹. Il existe deux bibliothèques de cartographies sous python permettant de réaliser des cartes de très grande qualité ainsi qu'une bibliothèque qui permet de faire des cartes web **basemap** et **cartopy**

basemap est une bibliothèque graphique qui permet avec l'aide de la bibliothèque **matplotlib** de faire des cartes. Chaque carte produite par l'intermédiaire de **basemap** est une figure de **matplotlib**. Ceci présente l'avantage de permettre d'accéder aux fonctionnalités avancées de **matplotlib** et d'obtenir des résultats de grande qualité en python. On peut en outre l'interfacier avec les systèmes de gestion de données comme MariaDB (MySQL) que nous utiliserons en première partie.

cartopy est une seconde bibliothèque, plus récente, qui pousse la logique d'intégration des cartes dans les figures de **matplotlib** jusqu'au bout en redéfinissant les axes d'une figure sous la forme de **GeoAxes**. Elle a vocation à remplacer **basemap** qui n'est plus maintenue. **cartopy** est une bibliothèque récente qui ne présente pas encore toutes les fonctionnalités de **basemap**. Elle présente cependant des avantages très importants qui font que, sauf cas particuliers, elle est plus simple d'utilisation tout en donnant d'aussi bons résultats. C'est cette bibliothèque que nous utiliserons dans ce cours.

Ce cours reprend, en les adaptant à nos besoins diverses sources (d'autres liens sont donnés en annexe) :

1. Le site de **cartopy**².
2. Divers tutoriels portant sur la bibliothèque GDAL³ qui est au coeur de toutes les applications de SIG gratuites ou payantes aujourd'hui.

Les bibliothèques nécessaires : **cartopy**, **pyshp**, **pyproj**, **shapely**, **geopandas**. Tous les tutoriels en ligne expliquent comment installer ces bibliothèques, je n'insiste donc pas. **cartopy** est dorénavant installé de façon standard dans les distributions de Python 3 notamment Anaconda.

1. Message du professeur Gayer : «le shapefile est au Raster ce qu'Illustrator (Inkscape) est à Photoshop (Gimp). Sous Gimp ou Photoshop, on manipule des images, qui sont des matrices (vu l'année dernière) ou encore bitmap. Sous Illustrator ou Inkscape on manipule des objets et on fait du dessin vectoriel. Les shapefiles sont l'équivalent de ces objets vectoriels avec des informations géographiques associées, et les Rasters (MNT) sont des images/matrices (vu l'année dernière) avec des informations géographiques associées.».

2. <https://scitools.org.uk/cartopy/docs/latest/>

3. <https://pcjericks.github.io/py-gdalogr-cookbook/>

Chapitre 8

Les fonds de cartes

Le fond de carte constitue le cadre sur lequel vont être projetées les données que l'on souhaite représenter. Ce peut être un simple cadre, une grille, une carte du monde ou une image. La librairie `cartopy` dispose de fonctions qui permettent de préparer ce fond avant l'ajout de données externes. L'établissement d'un fond nous permettra donc d'apprendre à utiliser ces fonctions.

8.1 Première carte

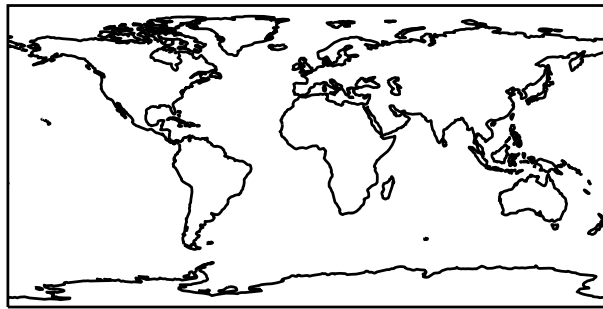


FIGURE 8.1 – Tracé des côtes (projection Plate Carrée)

`cartopy` est une librairie qui permet l'intégration d'une carte au sein d'un objet figure. Il suffit de déclarer que votre figure présente une projection cartographique pour que celle-ci «devienne» une carte et qu'elle présente des fonctionnalités supplémentaires. La carte produite figure 8.1 donne le résultat.

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 cm_1.py
5
6 simple first map (taken from cartopy tutorial)
7 """
8
9 import cartopy.crs as ccrs
10 import matplotlib.pyplot as plt
11
12 fig = plt.figure(figsize=(10/2.54,5/2.54))
13 # The great interest with cartopy is its ability to be intergrated into the pyplot
   environnement
14 ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
15
16 #add coastlines
17 ax.coastlines()
18
19 plt.savefig('../figures/cm_1.pdf',bbox_inches='tight')
20 plt.show()
```

8.1.1 Commentaires

`cartopy` introduit un argument de projection dans la définition de votre figure qui permet à `matplotlib` de savoir que cette figure est une carte.

Nous faisons appel à la fonction `coastlines()` qui comme vous vous en doutez est à l'origine du trait de côtes.

8.2 Coordonnées, système de référence et projections

Toute donnée qui peut-être positionnée dans l'espace peut ensuite être projetée sur un plan. Pour cela on doit pouvoir localiser la donnée dans un système de coordonnées attaché à un référentiel. La particularité qui nous intéresse ici est que nos données sont sur une planète (la terre en l'occurrence).

Le système de coordonnées le plus approprié est sphérique. Dans ce système les coordonnées d'un point sont définies par deux angles et un rayon (ici une altitude). Les angles sont la latitude (angle compté vers le nord à partir de l'équateur) et la longitude (angle compté vers l'est à partir d'un méridien — grand cercle passant par les pôles—, de référence, le méridien de Greenwich).

Comme la terre n'est pas totalement sphérique on utilise un ellipsoïde afin d'améliorer le positionnement précis d'un point sur le globe. Le globe n'étant pas non plus totalement ellipsoïdal des variations locales de forme de l'ellipsoïde sont introduites afin de mieux coller au géoïde que vous connaissez et qui est quasiment confondu avec le niveau des océans. On définit ainsi un référentiel plus élaboré que la sphère appelé un datum en SIG. WGS84 (World Geodetic System 1984) en est un mais il y en a d'autres (NAD27 and NAD83 utilisés aux USA par exemple)

La majorité des informations géographiques sont représentées sur des cartes. Les points sont projetés sur un plan en utilisant différents types de projections. Souvent, nous le verrons par la suite, les données que vous obtenez ne sont pas des données brutes du système sphérique mais des coordonnées déjà projetées. Pour représenter vos données et les comparer les unes avec les autres, les mélanger, vous devez donc connaître leur systèmes de coordonnées ainsi que leur système de référence.

8.2.1 Sélectionner une projection

Dans `cartopy` la projection est passée comme argument de la figure et les paramètres de la projection sont passés comme argument de celle-ci.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 bm_cm_2.py
5
6 Change basemap projection
7 """
8 from mpl_toolkits.basemap import Basemap
9 import cartopy.crs as ccrs
10 import matplotlib.pyplot as plt
11
12 fig=plt.figure(figsize=(15/2.54,7/2.54))
13
14 #basemap
15 ax = fig.add_subplot(121)
16 map = Basemap(projection = 'moll', lon_0 = 0)
17 map.drawcoastlines()
18 ax.set_title('$lon_0=0$')
19
20 #cartopy
21 ax2 = fig.add_subplot(122, projection=ccrs.Mollweide(central_longitude=180))
22 ax2.coastlines()
23 ax2.set_title('$lon_0=180$')
24
25 plt.savefig('../figures/bm_cm_2.pdf',bbox_inches='tight')
26 plt.show()

```

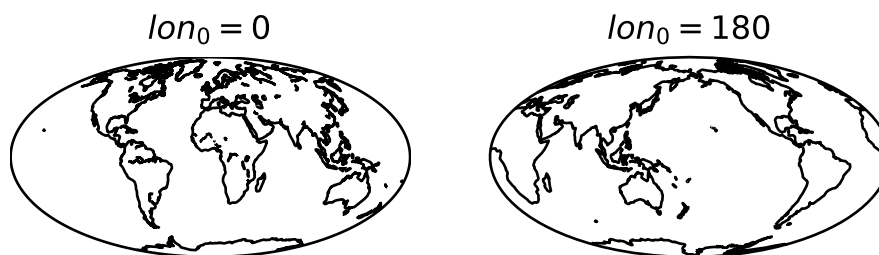


FIGURE 8.2 – Projection de Mollweide centrée sur le méridien de Greenwich à gauche et sur 180° à droite.

8.3 Les limites de la carte

Dans `cartopy` les limites de la carte sont décidées soit de façon automatique par `matplotlib` en fonction des données que vous représenterez, soit par des fonctions de la classe `axis` que vous utilisez (sans le savoir souvent) quand vous déclarez votre `subplot`. Si vous souhaitez définir des limites vous utiliserez les méthodes `set_extent(x1,y1,x2,y2)` ou `set_xlim(x1,y1)` et `set_ylim(y1,y2)`

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 bm_cm_4.py
5
6 Map limits
7 """
8
9 from mpl_toolkits.basemap import Basemap
10 import cartopy.crs as ccrs
11 import matplotlib.pyplot as plt
12
13
14 fig=plt.figure(figsize=(15/2.54,7/2.54))
15 #basemap
16 ax0=fig.add_subplot(121)
17 map = Basemap(llcrnrlon = -5, llcrnrlat = 42, urcrnrlon = 9., urcrnrlat = 52.,
18             projection = 'tmerc', lat_0 = 46, lon_0 = 2.5)
19 map.drawcoastlines()
20
21 #cartopy
22 ax1=fig.add_subplot(122, projection = ccrs.TransverseMercator(central_longitude=2.5,
23             central_latitude=46))
24 ax1.set_extent((-5,9,42,52))
25 ax1.coastlines()
26 ax1.gridlines()
27 plt.savefig('../figures/bm_cm_4.pdf',bbox_inches='tight')
28 plt.show()

```

Enfin vous noterez qu'à cette échelle le trait de côtes est assez grossier. Nous reviendrons sur ce point plus tard.

8.4 Echelle

Pas de carte sans échelle s'il vous plaît !

`cartopy` simplifie la gestion des limites mais la définition de l'échelle reste encore balbutiante à cause des limites dans le tracé des grilles. Il n'est pas possible par exemple de représenter les coordonnées sur la carte faite en transverse Mercator précédemment. SI CELA A CHANGÉ

Les figures 8.4, 8.5 et 8.6 et les scripts correspondant montrent les limites de la représentation de grilles. En particulier les labels de coordonnées ne sont disponibles qu'en Plate Carrée. Il faut donc soit donner la distance entre parallèles et méridiens dans la légende (figures 8.4), soit tout faire à la main (figure 8.6). Dans

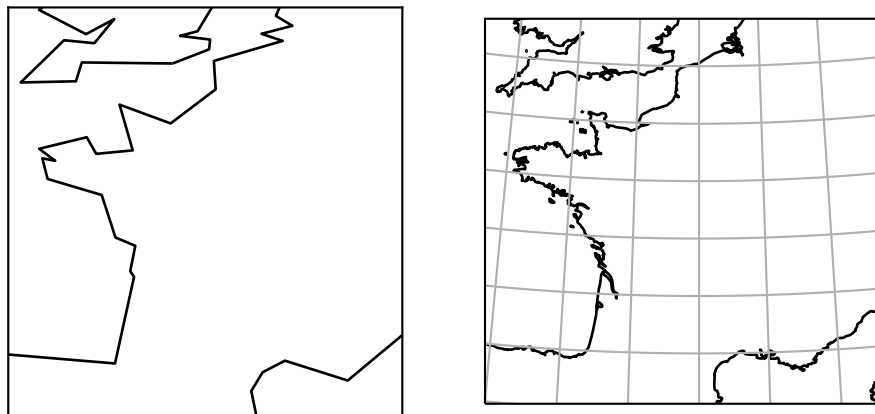


FIGURE 8.3 – Projection transverse Mercator. à gauche basemap, à droite cartopy.

cette dernière hypothèse, vous verrez en consultant le script qu'il est beaucoup plus simple de faire une échelle de longueur de que de représenter de façon fastidieuse toutes les longitudes/latitudes à la main !

```

1  #!/usr/bin/env python3
2  # -*- coding utf-8 -*-
3  """
4  cm_2.py
5
6  Change projection and add gridlines
7  """
8
9  import matplotlib.pyplot as plt
10 import cartopy.crs as ccrs
11 import matplotlib.ticker as mticker
12 from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
13 from matplotlib import rcParams
14
15 rcParams['font.size']=6
16
17 fig = plt.figure(figsize=(15/2.54,10/2.54))
18
19 # a first projection
20 ax1 = fig.add_subplot(211, projection=ccrs.Mercator(
21     central_longitude=0))
22
23 ax1.coastlines(resolution='110m') # resolutions available 50m and 10m
24
25 gl = ax1.gridlines(draw_labels=True)
26
27
28
29
30 #a second projection
31 ax2 = fig.add_subplot(212, projection=ccrs.Mollweide(
32     central_longitude=180))
33
34 # features can be used to include different kinds of back_grounds
35 ax2.coastlines(resolution='110m')
36 ax2.gridlines()
37
38
39
40 plt.savefig('../figures/cm_2.pdf')
41 plt.show()

```

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-

```

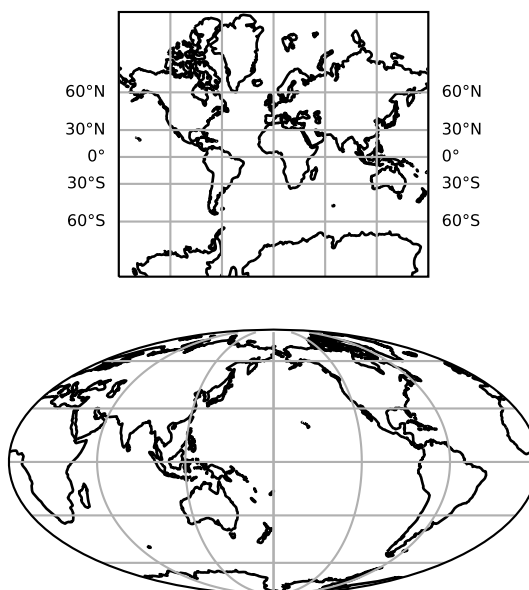


FIGURE 8.4 – Grilles en Plate Carrée et Mollweide, la distance entre méridiens est de 40° et entre parallèle de 30°

```

3  """
4  cm_2_2.py
5
6  Change gridlines and labels
7  at present it is less customizable than basemap but we can still do things
8  """
9
10 import matplotlib.pyplot as plt
11 import cartopy.crs as ccrs
12 from matplotlib import rcParams
13
14 rcParams['font.size'] = 6
15
16 fig = plt.figure(figsize=(16/2.54, 12/2.54))
17
18 myproj = ccrs.PlateCarree()
19
20 ax1 = fig.add_subplot(221, projection=myproj)
21
22 ax1.coastlines(resolution='110m') # resolutions available 50m and 10m
23 ax1.gridlines()
24 # problem with gridline ticks available in PlateCarree and Mercator only
25 # and tick overlaps use both xlocs and set_xticks same for y
26
27
28 ax2 = fig.add_subplot(222, projection=myproj)
29 ax2.coastlines(resolution='110m') # resolutions available 50m and 10m
30 ax2.gridlines(draw_labels=True, xlocs = range(-180,181,40), ylocs = range(-90,91,30))
31     # problems of labels to be solved ...
32
33 ax3 = fig.add_subplot(223, projection=myproj)
34 ax3.coastlines(resolution='110m') # resolutions available 50m and 10m
35 gl = ax3.gridlines(xlocs = range(-180,181,40), ylocs = range(-90,91,30)) # problems
36     of labels to be solved ...
37 gl.xlabels_bottom=True

```

```

36 gl.ylabels_left = True
37
38
39 ax4 = fig.add_subplot(224, projection=myproj)
40 ax4.coastlines(resolution='110m') # resolutions available 50m and 10m
41 gl2 = ax4.gridlines(xlocs = range(-180,181,40), ylocs = range(-90,91,30), color='
    darkblue', linestyle='--') # problems of labels to be solved ...
42 ax4.set_xticks(range(-180,181,40), crs=ccrs.PlateCarree(central_longitude=0))
43 ax4.set_yticks(range(-90,91,30), crs=ccrs.PlateCarree(central_longitude=0))
44
45
46 plt.savefig('../figures/cm_2_2.pdf',bbox_inches='tight')
47 plt.show()

```

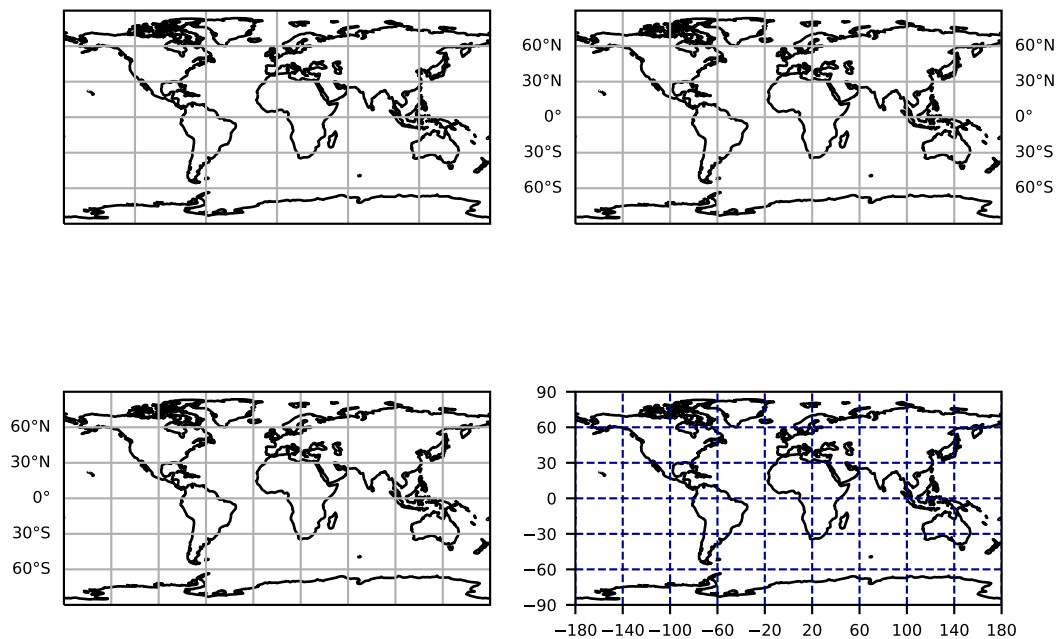


FIGURE 8.5 – Personnalisation de grilles en Plate Carrée

```

1 #!/usr/bin/env python 3
2 # -*- coding: utf-8 -*-
3 """
4 cm_4.py
5
6 Map limits
7 here again the limit of cartopy lies in the absence
8 of labels for lat and lon in this projection style
9 """
10
11 import matplotlib.pyplot as plt
12 import cartopy.crs as ccrs
13 from matplotlib.path import Path
14 from matplotlib import rcParams
15 import numpy as np
16
17 rcParams['font.size']=8
18
19
20 fig = plt.figure(figsize=(10/2.54,10/2.54))
21 # Define the projection
22 proj = ccrs.LambertConformal(central_longitude=7.5, central_latitude=46.0,
    standard_parallels = (43,49))
23
24 #define axis

```

```

25 ax = fig.add_subplot(111, projection = proj)
26 # everything is in this line !
27 ax.set_extent((-5,9,42,52))
28
29
30 ax.coastlines(resolution='10m') # best possible resolution
31 ax.gridlines(color = 'gray', linestyle = '-.-', xlocs= range(-7,11,2) , ylocs = range
    (40,56,2))
32
33
34 # a work around for labels not really nice but it works
35 ax.text(-4.1,42.1, '42N', ha='center', transform = ccrs.Geodetic())
36 ax.text(-4.3,44.1, '44N', ha='center', transform = ccrs.Geodetic())
37 ax.text(-4.8,46.1, '46N', ha='center', transform = ccrs.Geodetic())
38 ax.text(-5.3,48.1, '48N', ha='center', transform = ccrs.Geodetic())
39 ax.text(-5.6,50.1, '50N', ha='center', transform = ccrs.Geodetic())
40
41 ax.text(-3,41.7, '3W', ha='center', transform = ccrs.Geodetic())
42 ax.text(-1,41.85, '1W', ha='center', transform = ccrs.Geodetic())
43 ax.text(1,42, '1E', ha='center', transform = ccrs.Geodetic())
44 ax.text(3,42.1, '3E', ha='center', transform = ccrs.Geodetic())
45 ax.text(5,42.2, '5E', ha='center', transform = ccrs.Geodetic())
46 ax.text(7,42.2, '7E', ha='center', transform = ccrs.Geodetic())
47
48 #more simple draw a scale bar!
49 x1,y1 = (5,50)
50 deglat=1.65*60*np.cos(y1*np.pi/180)
51 x2=x1+100/deglat # put the second point 100k apart from the first one
52
53 ax.plot([x1,x2],[y1,y1], 'r-|', linewidth=2, transform=ccrs.PlateCarree())
54 ax.text(0.5*(x1+x2),y1+0.2, "100 km", color='r', ha='center', transform=ccrs.
    PlateCarree())
55
56
57
58 plt.savefig('../figures/cm_4.pdf',bbox_inches='tight')
59
60 plt.show()

```

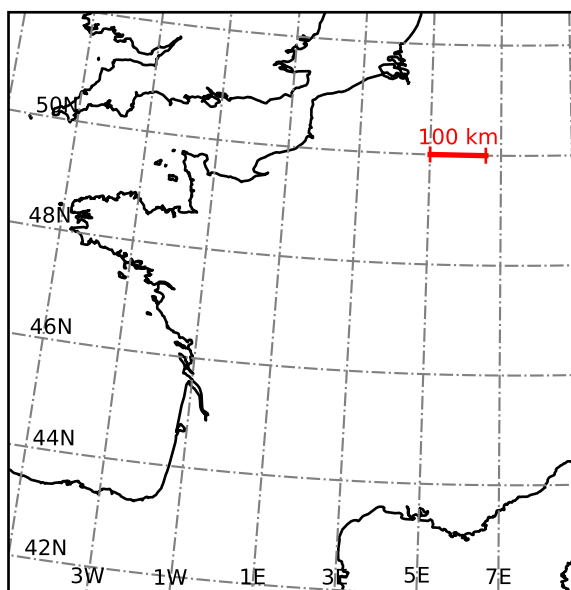


FIGURE 8.6 – Grilles, coordonnées et échelle en Lambert conforme

8.5 Les fonds de carte disponibles

Deux types de données sont disponibles de façon native dans chacun des bibliothèques pour faire des fonds de carte : des données vectorielles, traits de côtes ou rivières par exemple, et des données raster, les images topographiques ou satellitaires par exemple. Nous avons déjà utilisé les fonctions concernant le trait de côtes mais allons les passer en revue un peu plus en détail

8.5.1 Données vectorielles

`cartopy` présente peu de fonds de carte pré-installés. Deux fonds vectoriels sont disponibles, celui de la base de données [Natural Earth](#) et celui de la base [GSHHS](#) (plus ancien, utilisé par `pscoast` pour ceux qui se souviennent).

```

1 # -*- coding: utf-8 -*-
2
3 import matplotlib.pyplot as plt
4 from matplotlib import rcParams
5 import cartopy.crs as ccrs
6 import cartopy.feature as ft
7
8
9 rcParams['font.size']=6
10
11 fig=plt.figure(figsize=(15/2.54,15/2.54))
12
13
14 ax=fig.add_subplot(211,projection=ccrs.PlateCarree())
15 #in the case of world maps you can use the predefined datasets of NaturalEarth
16 ax.add_feature(ft.LAND)
17 ax.add_feature(ft.OCEAN)
18 ax.add_feature(ft.LAKES)
19 ax.add_feature(ft.RIVERS)
20 ax.add_feature(ft.BORDERS, linestyle='--', edgecolor='gray', linewidth=0.5)
21
22 ax.set_title('Natural_Earth_110m')
23
24 ax2=fig.add_subplot(212,projection=ccrs.PlateCarree())
25 ax2.set_xlim(-10,50)
26 ax2.set_ylim(30,60)
27
28 # in the case of 50 and 10m datasets you have to define the features from the
   NaturalEarthFeature
29 land_50m = ft.NaturalEarthFeature('physical', 'land', '50m', edgecolor='k', facecolor
   =ft.COLORS['land'])
30 ocean_50m = ft.NaturalEarthFeature('physical', 'ocean', '50m', edgecolor='face',
   facecolor=ft.COLORS['water'])
31 lakes_50m = ft.NaturalEarthFeature('physical', 'lakes', '50m', edgecolor='face',
   facecolor=ft.COLORS['water'])
32 rivers_50m = ft.NaturalEarthFeature('physical', 'rivers_lake_centerlines', '50m',
   edgecolor=ft.COLORS['water'], facecolor='none')
33
34 ax2.add_feature(land_50m)
35 ax2.add_feature(ocean_50m)
36 ax2.add_feature(lakes_50m)
37 ax2.add_feature(rivers_50m)
38
39 ax2.set_title('Natural_Earth_50m')
40
41
42 plt.savefig('../figures/cm5_backgrounds_vec.pdf', bbox_inches='tight')
43 plt.show()

```

Natural Earth 110m



Natural Earth 50m



FIGURE 8.7 – Fonds de carte disponibles dans `cartopy`

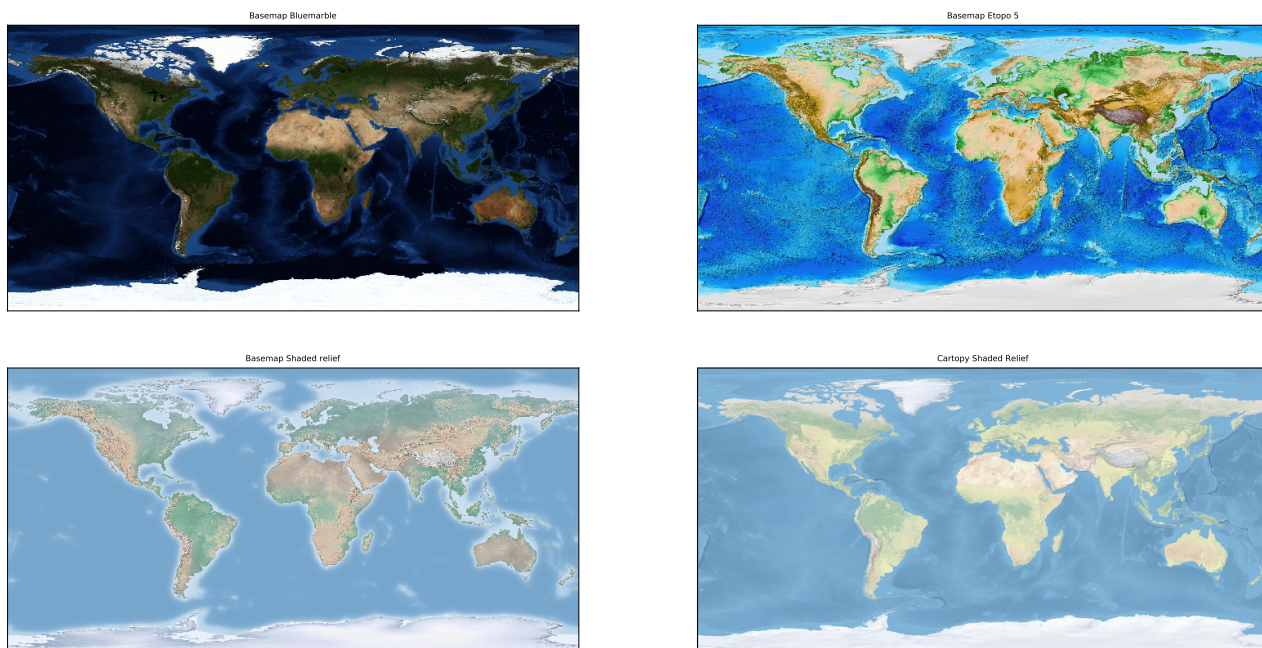


FIGURE 8.8 – Fonds de carte disponibles dans basemap et cartopy. À REFAIRE SANS BASEMAP

8.5.2 Images et fonds topographiques

`cartopy` ne propose qu'un fond de carte nommé `stock_img` qui correspond au shaded relief de Natural Earth. le script suivant montre à quoi ressemblent ces fonds de carte et la façon de les inclure dans votre figure.

```

1 # -*- coding: utf-8 -*-
2
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.basemap import Basemap
5 from matplotlib import rcParams
6 import cartopy.crs as ccrs
7
8 rcParams['font.size']=6
9
10
11 fig = plt.figure(figsize=(20,10))
12 ax1=fig.add_subplot(221)
13 maCarte=Basemap(ax=ax1)
14 maCarte.bluemarble(scale=0.5)
15 ax1.set_title('Basemap_ Bluemarble')
16
17 ax2=fig.add_subplot(222)
18 maCarte=Basemap(ax=ax2)
19 maCarte.etopo(scale=0.5)
20 ax2.set_title('Basemap_ Etopo_ 5')
21
22 ax3=fig.add_subplot(223)
23 maCarte=Basemap(ax=ax3)
24 maCarte.shadedrelief(scale=0.5)
25 ax3.set_title('Basemap_ Shaded_ relief')
26
27 ax4=fig.add_subplot(224, projection=ccrs.PlateCarree())
28 ax4.stock_img()
29 ax4.set_title('Cartopy_ Shaded_ Relief')
30
31 plt.savefig('../figures/bm5_backgrounds_im.pdf', bbox_inches='tight')
32
33 plt.show()

```

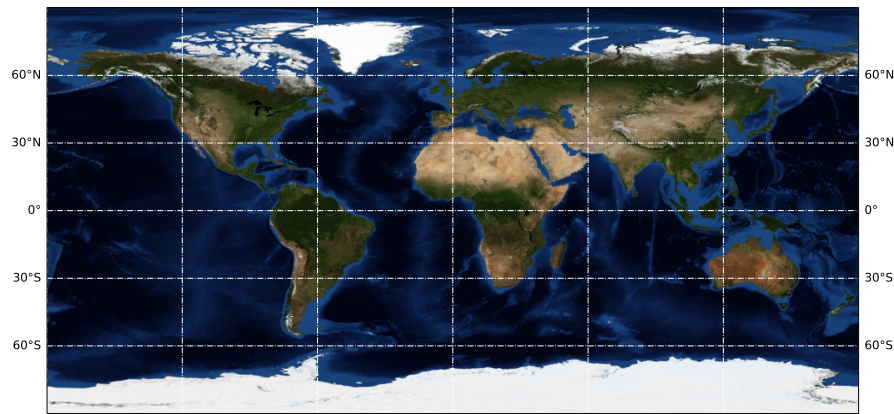


FIGURE 8.9 – Image Blue Marble incluse comme fond de carte dans cartopy.

8.5.3 Images de fond

Je viens de l'écrire, `cartopy` n'a qu'un nombre limité d'images de fond de carte préchargées. Par contre il est relativement simple d'ajouter une image en utilisant une méthode appelée `background_img()`

```

1 # -*- coding: utf-8 -*-
2
3 import matplotlib.pyplot as plt
4 from mpl_toolkits.basemap import Basemap
5 from matplotlib import rcParams
6 import cartopy.crs as ccrs
7
8 rcParams['font.size']=6
9
10
11 fig = plt.figure(figsize=(20,10))
12 ax1=fig.add_subplot(221)
13 maCarte=Basemap(ax=ax1)
14 maCarte.bluemarble(scale=0.5)
15 ax1.set_title('Basemap_□Bluemarble')
16
17 ax2=fig.add_subplot(222)
18 maCarte=Basemap(ax=ax2)
19 maCarte.etopo(scale=0.5)
20 ax2.set_title('Basemap_□Etopo_□5')
21
22 ax3=fig.add_subplot(223)
23 maCarte=Basemap(ax=ax3)
24 maCarte.shadedrelief(scale=0.5)
25 ax3.set_title('Basemap_□Shaded_□relief')
26
27 ax4=fig.add_subplot(224, projection=ccrs.PlateCarree())
28 ax4.stock_img()
29 ax4.set_title('Cartopy_□Shaded_□Relief')
30
31 plt.savefig('../figures/bm5_backgrounds_im.pdf', bbox_inches='tight')
32
33 plt.show()

```

Les possibilités augmentent donc rapidement. Nous allons maintenant accéder aux serveurs de tuiles qui offrent de nouvelles possibilités et seront très utiles lorsque nous nous intéresseront aux cartes destinées à internet.

8.5.4 les serveurs de tuiles wms et wmts

Les serveurs de données géographiques (web map services) ou de tuiles (web map tile services) sont des sites web offrant la possibilité par le biais de la connexion à une URL de récupérer en temps réel des images et des

données vectorielles sur internet et de les représenter. Google maps et Google Earth sont des applications qui font appels aux serveurs de tuiles et de données de Google. L'immense avantage des serveurs de données et de tuiles réside dans leur capacité à s'adapter en temps réel aux requêtes qui leurs sont envoyées et en particulier à l'échelle de la carte. Ainsi, en zoomant et dé-zoomant, on envoie une requête au serveur qui en réponse augmente ou diminue la quantité de données téléchargées en fonction de l'emprise de la carte. Le désavantage de l'usage des WMS ou WMTS est la nécessité d'une connexion. `cartopy` permet l'intégration de données issues d'un serveur dans une figure matplotlib. Il permet en outre, fait nouveau, la mise à jour des données en temps réel. Quand on zoome la précision des données augmente jusqu'à atteindre la résolution maximale du serveur.

```

1  """
2  Use wmts servers using cartopy
3  """
4  import matplotlib.pyplot as plt
5  import matplotlib.path_effects as PathEffects
6  from owslib.wmts import WebMapTileService
7
8  import cartopy.crs as ccrs
9
10
11 def main():
12
13     # Plot setup
14     plot_CRS = ccrs.PlateCarree()
15
16     # URL of NASA GIBS
17     URL = 'http://gibs.earthdata.nasa.gov/wmts/epsg4326/best/wmts.cgi'
18     wmts = WebMapTileService(URL)
19
20
21     fig = plt.figure(figsize = (15, 22)) #must be reasonably big because of the
22         sampling of the image during savefig.
23     ax = fig.add_subplot(311, projection = plot_CRS)
24
25     # Blue Marble with bathymetry
26     layer = 'BlueMarble_ShadedRelief_Bathymetry'
27     ax.add_wmts(wmts, layer)
28     ax.gridlines(draw_labels = True, xlocs = range(-180, 181, 60), ylocs = range
29         (-90, 91, 30), color = 'w', linestyle = '-.')
30
31     ax2 = fig.add_subplot(312, projection = plot_CRS)
32
33     layer = 'ASTER_GDEM_Color_Shaded_Relief'
34     ax2.add_wmts(wmts, layer)
35     ax2.coastlines(resolution = '110m')
36     ax2.gridlines(draw_labels = True, xlocs = range(-180, 181, 60), ylocs = range
37         (-90, 91, 30), color = 'w', linestyle = '-.')
38
39     ax3 = fig.add_subplot(313, projection = plot_CRS)
40
41     layer = 'GPW_Population_Density_2020'
42     ax3.add_wmts(wmts, layer)
43     ax3.coastlines(resolution = '110m')
44     ax3.gridlines(draw_labels = True, xlocs = range(-180, 181, 60), ylocs = range
45         (-90, 91, 30), color = 'k', linestyle = '-.')
46
47     plt.savefig('../figures/cm_wmts_1.pdf', bbox_inches = 'tight')
48
49     plt.show()
50
51 def other_examples():
52
53     # Plot setup

```

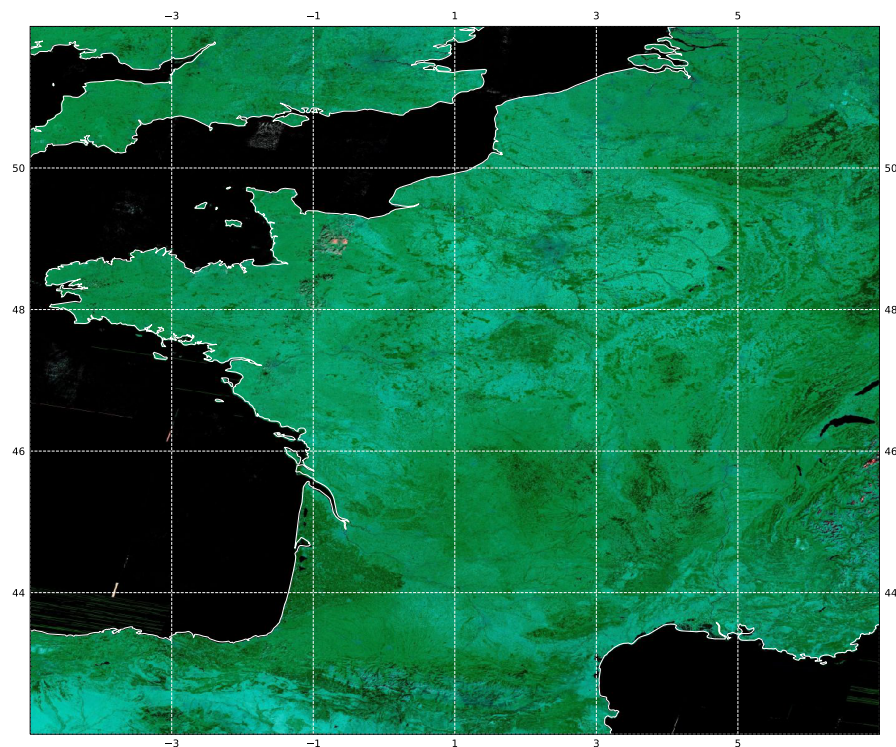


FIGURE 8.10 – Utilisation de serveurs de tuiles avec cartopy. Image Landsat ETM de la France, 16 août 1999

```

54     plot_CRS = ccrs.PlateCarree()
55
56     fig = plt.figure(figsize = (15, 15))
57     ax = fig.add_subplot(111, projection = plot_CRS)
58
59
60     # URL of NASA GIBS
61     URL = 'http://gibs.earthdata.nasa.gov/wmts/epsg4326/best/wmts.cgi'
62     wmts = WebMapTileService(URL)
63
64     # Layers for MODIS true color and snow RGB
65     layer = 'Landsat_WELD_CorrectedReflectance_Bands157_Global_Annual'
66
67     date_str = '1999-08-16'
68
69
70     ax.set_xlim(-5, 7)
71     ax.set_ylim(42, 52)
72
73     ax.add_wmts(wmts, layer, wmts_kwargs = {'time': date_str})
74
75     ax.coastlines(resolution = '10m', color = 'w')
76     ax.gridlines(draw_labels = True, xlocs = range(-5, 8, 2), ylocs = range(42,
77                                     53, 2), color = 'w', linestyle = '--')
78
79     plt.savefig('../figures/cm_France_Landsat.pdf', bbox_inches = 'tight')
80     plt.show()
81
82 if __name__ == '__main__':
83     main()
84     other_examples()

```

Vous avez donc dorénavant accès à des milliers de données que vous pouvez importer dans votre carte en un rien de temps. Il vous suffit de consulter les sites qui vous intéressent en particulier le site de la NASA dont je me suis servi et qui offre l'accès à pratiquement toute son imagerie spatiale :

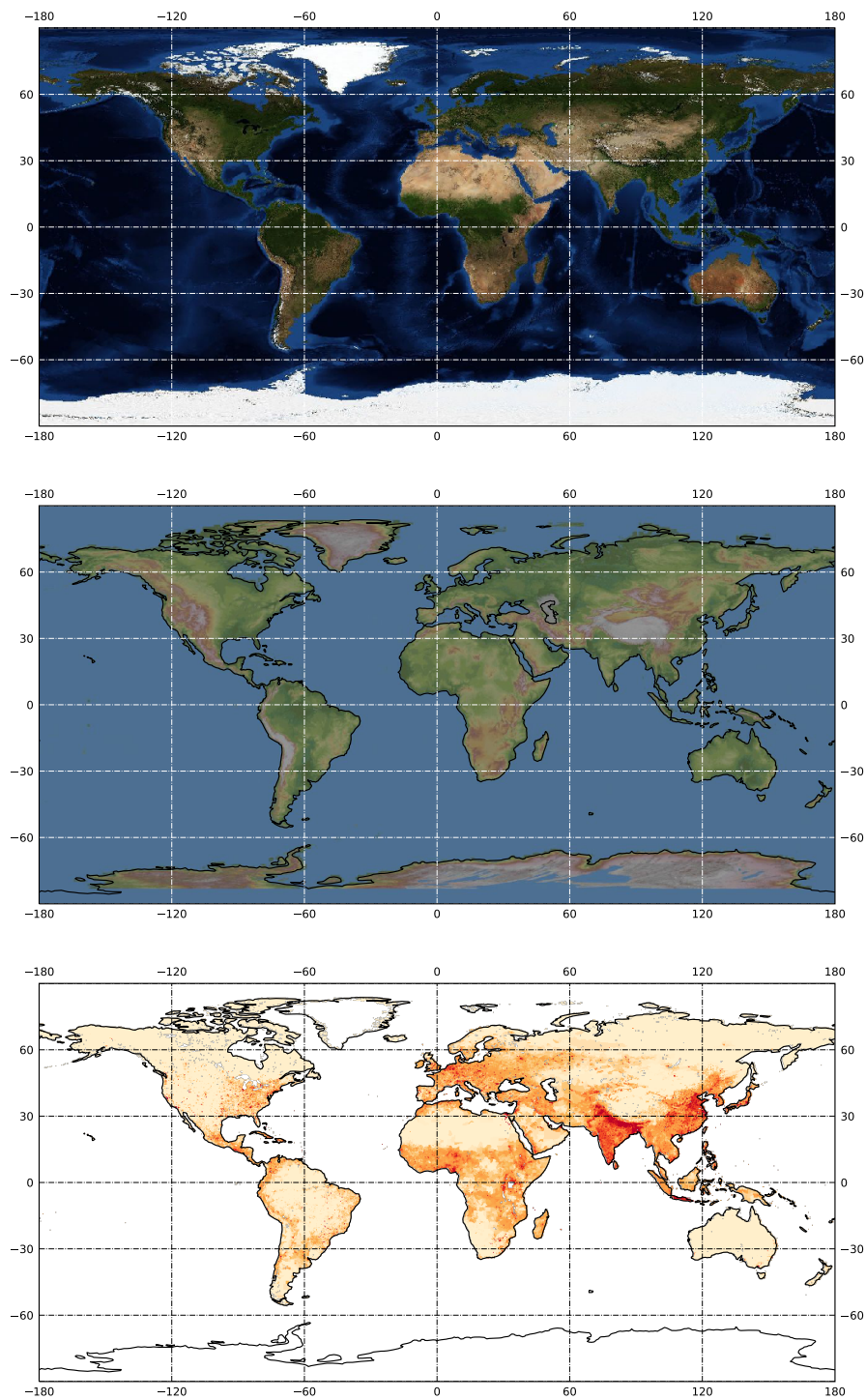


FIGURE 8.11 – Utilisation de serveurs de tuiles avec cartopy. De haut en bas : Blue Marble, ASTER GDEM, Densité de population 2020.

<https://wiki.earthdata.nasa.gov/display/GIBS/GIBS+Available+Imagery+Products>

Chapitre 9

Ajouter des données

9.1 Points et trajets

Nous allons commencer par ajouter des points avec `cartopy` afin de comprendre la différence entre les arguments `projection` et `transform`. Ajoutons donc un point à notre toute première figure afin de comprendre. Le script `cm_1_2.py` montre comment faire

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 cm_1_2.py
5
6 projection vs transform
7 """
8
9 import cartopy.crs as ccrs
10 import matplotlib.pyplot as plt
11
12 fig = plt.figure(figsize=(5,3))
13 # the great interest with cartopy is its ability to be
14 # intergrated into the pyplot environnement
15 ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
16
17 #this becomes necessary if you plot data that is not
18 # in the same projection
19 ax.set_xlim(-180,50)
20 ax.set_ylim(0,90)
21
22 #add coastlines
23 ax.coastlines()
24
25 #Point 1
26 UTM_X = 565718.
27 UTM_Y = 3980998.
28 ax.plot(UTM_X, UTM_Y, 'ro', transform = ccrs.UTM(11))
29
30 #Point 2
31 LON_X, LAT_X = (2.4, 48.8)
32 ax.plot(LON_X,LAT_X,'bo', transform=ccrs.Geodetic())
33
34
35 plt.savefig('../figures/OnePoint.pdf',bbox_inches='tight')
36
37
38 plt.show()
```

La figure 9.1 montre le résultat et l'on comprend bien la différence entre l'argument `projection` utilisé lors de la définition de la figure ou de l'axe et l'argument `transform` utilisé lors de la représentation d'un point. `projection` correspond à la projection de la carte que vous êtes en train de créer, `transform` correspond à la projection dans laquelle sont définis les points que vous voulez représenter. `cartopy` permet donc la conversion à la volée des systèmes de projection ce qui représente une amélioration considérable par rapport à `basemap`.

Notons que, par défaut, python fixe automatiquement des limites qui correspondent à peu de choses près aux points que vous avez définis (essayez pour voir en commentant les lignes `ax.set_xlim` et `ax.set_ylim`).



FIGURE 9.1 – Ajouter des points dans différentes projections

Des données en latitudes et longitudes peuvent être représentées en utilisant `transform=PlateCarree()` ou `transform=Geodetic()`. Attention cependant ; dans le premier cas il s'agit de coordonnées Longitudes/latitudes projetées sur un plan, dans le second il s'agit de coordonnées sphériques. Si nous représentons un trajet entre deux villes le résultat dépendra de ce choix.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  cm_1_2.py
5
6  projection vs transform
7  """
8
9  import cartopy.crs as ccrs
10 import matplotlib.pyplot as plt
11
12 fig = plt.figure(figsize=(5,3))
13 # great interest with cartopy is its ability to be
14 # integrated into the pyplot environnement
15 ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
16
17 # this becomes necessary if you plot data that is not in
18 # the same projection
19 ax.set_xlim(-100,30)
20 ax.set_ylim(0,90)
21
22 # add coastlines
23 ax.coastlines()
24
25 # Point 1
26 LON_P, LAT_P = (2.4, 48.8)
27 ax.plot(LON_P,LAT_P,'bo', transform=ccrs.Geodetic())
28
29 LON_NY, LAT_NY = -75, 43
30 ax.plot(LON_NY,LAT_NY,'ro', transform=ccrs.Geodetic())
31
32 ax.plot([LON_P,LON_NY],[LAT_P,LAT_NY],'--',transform=ccrs.PlateCarree())
33 ax.plot([LON_P,LON_NY],[LAT_P,LAT_NY],'-',transform=ccrs.Geodetic())
34
35 plt.savefig('../figures/OneTrajectory.pdf',bbox_inches='tight')
36
37
38 plt.show()

```

La figure 9.2 montre ainsi dans le premier cas python tracera une droite, dans le second un grand cercle qui correspond à la plus courte distance entre les deux points. Notons que pour représenter le grand cercle python discrétise la trajectoire¹.

1. Et là, à ma grande honte, j'avoue que je n'ai pas encore trouvé comment contrôler le nombre de points de discrétisation !

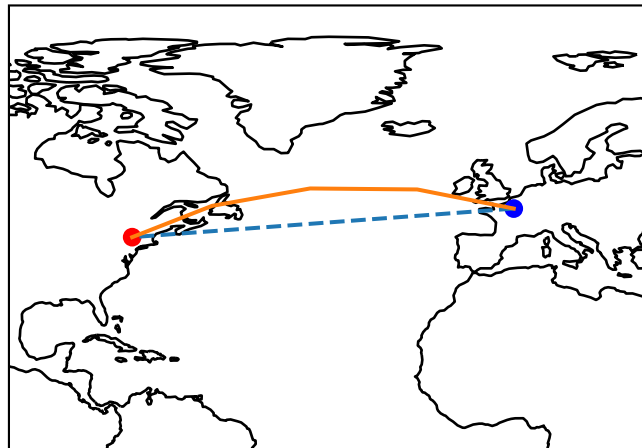


FIGURE 9.2 – Deux trajets

9.2 Deux exemples en Sciences de la terre

Ces deux exemples permettent de faire appel à la fonction `scatter` et à la représentation d'une matrice avec `contourf`

9.2.1 Carte de la sismicité

L'an passé E. Gayer vous a appris à importer des données dans `Pandas`, notamment un catalogue mondial de séismes pour le mois de février 2017. Puis vous avez téléchargé le contour des côtes et avez tracé une belle carte en projection plate carrée des continents et des séismes. Maintenant vous devriez donc être capable de reprendre votre catalogue et de faire la figure 9.3 (aux légendes près). Dans les deux cas les bibliothèques font appel à la fonction `scatter` de `matplotlib`. La différence réside essentiellement dans la façon de définir la carte et de projeter les données. Pour le plaisir je vous représente ces tremblements de terre dans deux projections différentes.

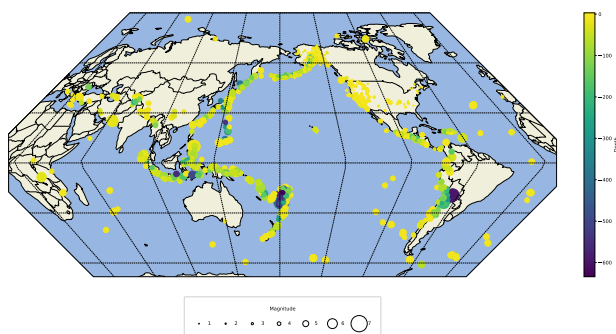


FIGURE 9.3 – Sismicité mondiale (Février 2017)

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  cm_5_EQ.py
5
6  Draw map of Feb 2017 Earthquakes
7  color = depth
8  size = magnitude
9
10 Original Idea borrowed from E. Gayer
11 """
12
13 import cartopy
14 import cartopy.crs as ccrs

```

```

15 import matplotlib.pyplot as plt
16 import pandas as pd
17 import numpy as np
18
19 from mpl_toolkits.axes_grid1.inset_locator import inset_axes
20
21 #Figure
22 fig = plt.figure(figsize = (20, 10))
23 ax = fig.add_subplot(111, projection=ccrs.EckertI(central_longitude=180))
24
25
26 ax.add_feature(cartopy.feature.OCEAN, zorder=0)
27 ax.add_feature(cartopy.feature.LAND, zorder=0, edgecolor='black')
28 ax.add_feature(cartopy.feature.BORDERS, zorder=0)
29
30 #~ ax.set_global()
31 #~ ax.gridlines(draw_labels=True, zorder=4, color='k', linestyle='--', ylocs=range
    (-90,91,30), xlocs=range(-180,181,40))
32 ax.gridlines(zorder=4, color='k', linestyle='--', ylocs=range(-90,91,30), xlocs=range
    (-180,181,40))
33
34
35
36
37 EQall_month = pd.read_csv('../include/txt/EQall_month.csv')
38 print(EQall_month['time'])
39
40 print(min(EQall_month['mag']), max(EQall_month['mag']))
41 x, y = EQall_month['longitude'].tolist(), EQall_month['latitude'].tolist()
42 sc = ax.scatter(x, y, s = np.exp(EQall_month['mag']), c = -EQall_month['depth'],
    cmap = 'viridis', zorder = 2, transform=ccrs.PlateCarree())
43
44 #####
45 # Tricky legend for magnitude AND depth
46 #     1) For depth we use a simple colorbar
47 #     2) For magnitude label won't work so we use a "phantom"
48 #     plot to build the legend of the scatter plot
49 #####
50 #color bar
51 axins = inset_axes(ax,
52     width="2%", # width = 5% of parent_bbox width
53     height="100%", # height : 50%
54     loc='lower_left',
55     bbox_to_anchor=(1.05, 0., 1, 1),
56     bbox_transform=ax.transAxes,
57     borderpad=0,
58     )
59
60 cbar = plt.colorbar(sc, cax=axins)
61 cbar.set_label('Depth')
62
63 #phantom plot
64 for i in range(7):
65     plt.scatter([], [], s = np.exp(i+1), edgecolor = 'k', facecolor = 'none',
66         label = str(i+1))
67
68 #legend
69 h, l = plt.gca().get_legend_handles_labels()
70 ax.legend(h, l, title = "Magnitude", labels spacing = 2, borderpad = 2, frameon = True
71     , numpoints = 1, ncol = 7, framealpha = 0.9, bbox_to_anchor = (0.5, -0.15),
72     loc = 'center')
73
74 plt.savefig('../figures/cm_5_EQ.pdf',bbox_inches='tight')
75 # See the result
76 plt.show()

```

9.2.2 USGS Earthquake catalogue

Evidemment l'idéal serait de pouvoir aller récupérer les données directement sur le site de l'USGS. Dont acte nous allons représenter les tremblements de terre de magnitude supérieure à 5 des derniers trente jours

```

1  """
2  Download the earthquake catalogue of USGS
3  get the M>=5 earthquakes of the last 30 days
4  and plot them
5  """
6
7  import requests
8  import json
9  import cartopy.crs as ccrs
10 import numpy as np
11 import matplotlib.pyplot as plt
12 from datetime import date, timedelta
13 from mpl_toolkits.axes_grid1.inset_locator import inset_axes
14
15
16
17
18 time_interval = 30 #decide the time interval (in days) you wish to search in the
    catalogue
19 today = date.today() #get today's date
20 startday = today - timedelta(days=time_interval) # get the starting date of seach
21
22
23
24 et=today.strftime("%Y-%m-%d")
25 st=startday.strftime("%Y-%m-%d")
26 minmag='5'
27
28 turl= "https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime=%s&
    endtime=%s&minmag=%s" % (st,et,minmag)
29
30 response = requests.get(turl)
31
32 data = response.json()
33
34 lat=[]
35 lon=[]
36 depth=[]
37 mag=[]
38
39 for f in data['features']:
40     mag.append( f['properties']['mag'] )
41     coord = f['geometry']['coordinates']
42     lon.append(coord[0])
43     lat.append(coord[1])
44     depth.append(coord[2])
45
46
47 fig = plt.figure(figsize = (15, 7))
48 ax = fig.add_subplot(111, projection = ccrs.PlateCarree(central_longitude=180))
49 ax.coastlines()
50 ax.gridlines(xlocs=range(-180,181,60),ylocs=range(-90,91,30))
51 ax.set_xticks(range(-180,181,60))
52 ax.set_yticks(range(-90,91,30))
53 ax.set_extent((-180,180,-90,90))
54
55 sc = ax.scatter(x = lon, y = lat, s = np.exp(np.array(mag)), c = -np.array(depth),
    transform=ccrs.Geodetic(), edgecolors='grey', zorder=4)
56
57 ax.set_title("Start:%s;┘End:%s" % (st,et))
58 #color bar
59 axins = inset_axes(ax,

```

```

60         width="2%", # width = 5% of parent_bbox width
61         height="100%", # height : 50%
62         loc='lower_left',
63         bbox_to_anchor=(1.05, 0., 1, 1),
64         bbox_transform=ax.transAxes,
65         borderpad=0,
66     )
67
68 cbar = plt.colorbar(sc, cax=axins)
69 cbar.set_label('Depth')
70
71
72 #phantom plot
73 for i in range(3):
74     plt.scatter([], [], s = np.exp(i+5), edgecolor = 'k', facecolor = 'none',
75               label = str(i+5))
76
77 #legend
78 h, l = plt.gca().get_legend_handles_labels()
79 ax.legend(h, l, title = "Magnitude", labelspace = 2, borderpad = 3, frameon = True
80         , numpoints = 1, ncol = 5, framealpha = 0.9, loc = 'lower_left')
81
82 plt.savefig('../figures/cm_EQ30.pdf', bbox_inches='tight')
83 plt.show()

```

cartopy

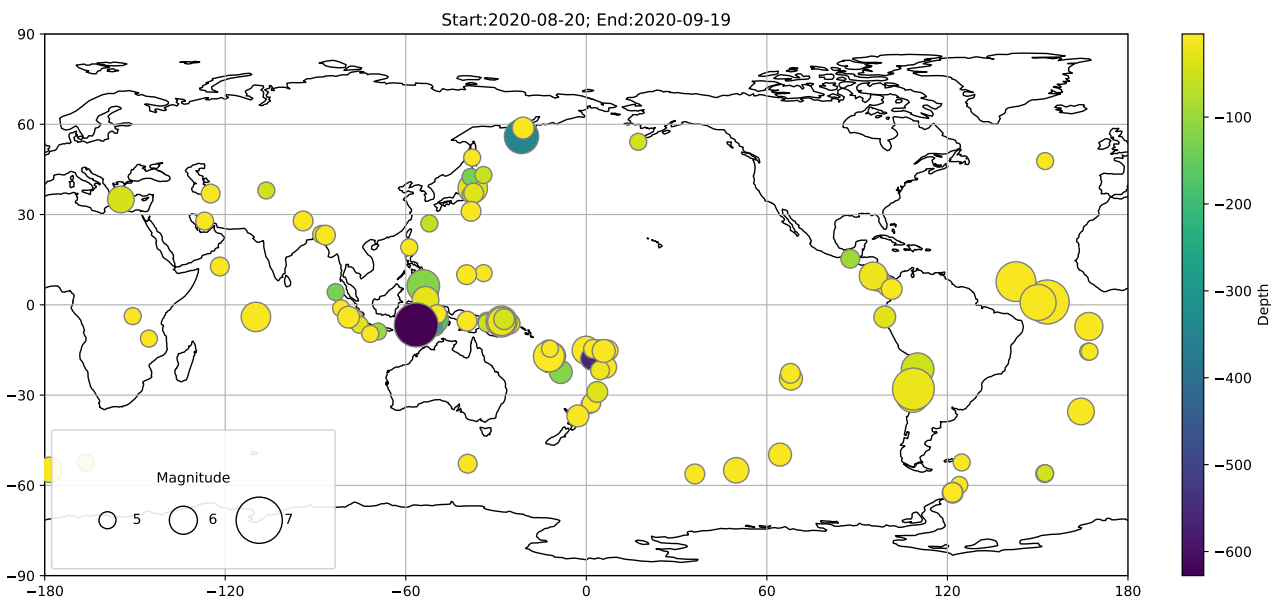


FIGURE 9.4 – Sismicité mondiale des trente dernier jours ($M \geq 5$). Exécuté le 19/09/2020

9.2.3 Variation temporelle du champ magnétique (contour plot)

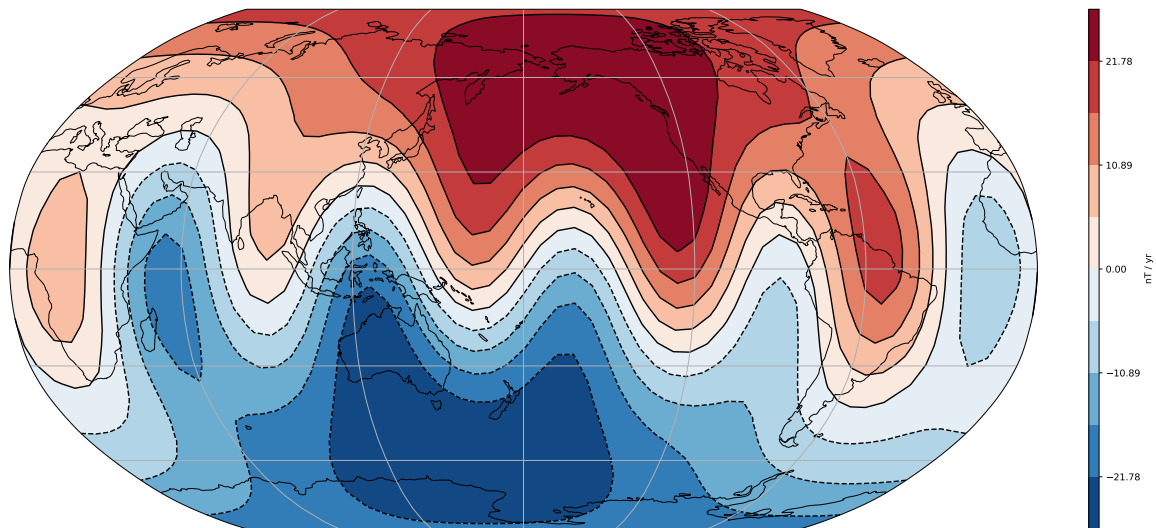
Là encore, vous devriez pouvoir vous en sortir. Le fichier `SVZ.txt` représente une approximation de la variation temporelle de la composante verticale Z du champ magnétique \mathbf{B}^2 . Les données correspondent à une grille de 61×61 qui s'étend des colatitudes 1 à 179 (du Nord au Sud) et des longitudes 0 à 360 (en comptant vers l'Est). L'objectif est donc de reproduire la figure 9.5

```

1 # -*- coding: utf-8 -*-
2 """
3 cm_5_SVZ.py

```

2. Merci à A. Fournier pour le programme (Python) permettant de calculer cette composante.

FIGURE 9.5 – Variation temporelle de la composante verticale du champs magnétique \dot{Z} .

```

4
5 Draw map of the time variation of the Z component of the magnetic field
6 Courtesy of A. Fournier
7
8 """
9
10 import numpy as np
11 import matplotlib.pyplot as plt
12 import cartopy.crs as ccrs
13
14 from mpl_toolkits.basemap import Basemap
15 from mpl_toolkits.axes_grid1.inset_locator import inset_axes
16
17 SVZ = np.loadtxt('../include/txt/SVZ.txt')
18
19 #size of the grid over the sphere
20 nlon = 61 # 61 points in longitude
21 nlat = 31 # 31 points in colatitude
22
23 #coordinates of SVZ points
24 colat = np.linspace(1., 179., nlat, endpoint = True) # we exclude the poles
25 longi = np.linspace(-180., 180., nlon, endpoint = True)
26
27 my_crs = ccrs.PlateCarree()
28 fig = plt.figure(figsize = (20, 10))
29 ax = fig.add_subplot(111, projection = ccrs.Robinson(central_longitude=180))
30
31 ax.coastlines()
32 #~ ax.gridlines(draw_labels=True, xlocs=range(-180,181,60), ylocs=range(-90,91,30))
33 ax.gridlines(xlocs=range(-180,181,60), ylocs=range(-90,91,30))
34
35
36 #transform coordinates and reshape SVZ into a 2D grid
37 latit = 90 - colat
38 lons, lats = np.meshgrid(longi, latit)
39 svz2d = np.reshape(SVZ, (nlat, nlon))
40
41 #define color levels
42 maxsvz = max(np.abs(SVZ))
43 clevs = np.linspace(-maxsvz, maxsvz, 11, endpoint = True)
44
45 #plot SVZ
46 cs1 = ax.contour(lons, lats, svz2d, clevs, colors = 'k', linewidth = 0.5,
47                 transform = my_crs)

```



```
47 cs2 = ax.contourf(lons, lats, svz2d, clevs, cmap = plt.cm.RdBu_r, transform =
    my_crs)
48
49 #add colorbar
50 axins = inset_axes(ax,
51     width="1%", # width = 5% of parent_bbox width
52     height="100%", # height : 50%
53     loc='lower_left',
54     bbox_to_anchor=(1.05, 0., 1, 1),
55     bbox_transform=ax.transAxes,
56     borderpad=0,
57     )
58
59 cb = plt.colorbar(cs2, cax=axins)
60 cb.set_label('nT/yr')
61
62 plt.savefig('../figures/cm_5_SVZ.pdf',bbox_inches='tight')
63 plt.show()
```

Chapitre 10

Pyproj

En python la librairie qui gère les transformations s'appelle `pyproj`. C'est comme beaucoup de librairies python un «wrapper» qui permet d'accéder à une librairie appelée PROJ.4 développée à l'origine par l'USGS. Voici un exemple de la façon dont `pyproj` permet de changer de coordonnées et de calculer des distances.

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 bm_3.py
5
6 Coordinate transformations using pyproj
7 """
8 #libray
9 import pyproj
10
11 #
12 # Calculate distance between two points on earth
13 #
14
15 wgs84_geod = pyproj.Geod(ellps = 'WGS84')
16 lat1, lon1 = (27.23150120120691, -91.522575364888709) # somewhere in the gulf of
17             mexico
18 lat2, lon2 = (27.239416154284083, -91.520817319790083) # near lat1, lon1
19 az12, az21, dist = wgs84_geod.inv(lon1, lat1, lon2, lat2)
20 print("Azimuths from point 1 to point 2 (and vice versa): %f°, %f°\nDistance between
21       points: %f km" % (az12, az21, dist))
22
23 #
24 # Change projection
25 #
26 # Define original coordinates
27 UTM_X = 565718.
28 UTM_Y = 3980998.
29
30 # Define source and destination projections
31 srcProj = pyproj.Proj(proj = "utm", zone = "11", ellps = "clrk66", units = "m")
32 dstProj = pyproj.Proj(proj = "longlat", ellps = "WGS84", datum = "WGS84")
33
34 # Do the coordinate transformation
35 Long, Lat = pyproj.transform(srcProj, dstProj, UTM_X, UTM_Y)
36 print("Coordinate transformation from UTM 11 zone\n\t%0.4f, %0.4f => %0.4fE,
37       %0.4fN" % (UTM_X, UTM_Y, Long, Lat))
38
39 #
40 # Some calculations
41 #
42 # Add ten kilometers to one UTM coordinate
43 #
```

```
44 UTM_Y += 10000
45 Long2, Lat2 = pyproj.transform(srcProj, dstProj, UTM_X, UTM_Y)
46 print("Coordinate transformation from UTM 11 zone\n\t%.4f, %.4f => %.4f E, %.4f
      f N"      % (UTM_X, UTM_Y, Long2, Lat2))
47
48 #
49 # Find de position of a point located 10km north of the original point
50 #
51 azimuth = 360
52 distance = 10000
53
54 geod = pyproj.Geod(ellps = "WGS84") # instantiate the Geod class to do the
      calculation
55 Long3, Lat3, invangle = geod.fwd(Long, Lat, azimuth, distance)
56 print("The point located 10km to the North of %.4f E, %.4f N has longitude %.4f
      E and latitude %.4f N"      % (Long, Lat, Long3, Lat3))
```

vous devez voir s'afficher ce qui suit à l'écran

```
Coordinate transformation from UTM 11 zone
565718.0000,3980998.0000 => -116.2711 E,35.9730 N
Coordinate transformation from UTM 11 zone
565718.0000,3990998.0000 => -116.2703 E,36.0631 N
The point located 10km to the North of -116.2711 E,35.9730 N has longitude -116.2711 E
and latitude 36.0631 N
```

Les deux classes de pyproj sont utilisées dans ce script : Proj et Geod. La première gère les transformations, la seconde gère les calculs le long de grand cercles. Nous verrons diverses utilisation de cette librairie par la suite notamment quand il s'agit de définir les limites de cartes dans différentes projections.

Chapitre 11

Les shapefiles

11.1 Formes et géométries

Dans un SIG, les positions dans l'espace sont représentées par ce qu'on appelle une géométrie (geometry). il peut s'agir d'un point, d'une ligne, d'un polygone ou de toute combinaison de ces objets de base. Une géométrie en elle-même n'est rien d'autre qu'une forme (shape). Dans le SIG, cette forme est associée à des attributs. Un polygone peut par exemple représenter un arrêt de bus ou le bâtiment de l'IPGP, une ligne, une route, un point lumineux, etc.

L'association d'une géométrie avec des attributs s'appelle une entité/caractéristique (feature) et c'est l'unité de base de la programmation d'un SIG.

Une géométrie est appelée une donnée vectorielle car elle est composée d'une succession de points quelconques définis par leurs coordonnées dans l'espace. En raison de son importance, nous détaillons ci-après le modèle de données couramment utilisé pour travailler avec des formes et des géométries

11.2 OGR Data model

Comme nous l'avons vu dans ce cours, les ensembles de données doivent être organisés de manière rationnelle afin que chacun puisse y avoir accès. Un modèle de données est destiné à définir la manière dont un type de données doit être organisé. Le modèle de données vectorielles présenté et utilisé ici est celui initialement développé par le consortium Open GIS dans une bibliothèque appelée OGR¹. Il est compatible avec des dizaines de sources de données (formats de fichier et de base de données). Pour les habitués des bases de données que vous êtes devenus ce sera donc simple car l'analogie avec une base faite de tables liées entre elles est évidente (figure 11.1).

Une source de données dans ce modèle contient une ou plusieurs couches. Chaque couche possède une référence spatiale unique (modèle de référence et projection cartographique). Chaque couche possède en outre des entités. Chaque entité contient des géométries et des attributs. La bibliothèque `python` qui traite des jeux de données compatibles OGR s'appelle `osgeo.ogr`. Nous allons l'utiliser dans l'exemple ci-dessous.

11.3 Shapefile structure

Les types de fichiers les plus utilisés pour stocker des données vectorielles sont ce que l'on appelle des fichiers de formes (shapefile, extension `.shp`). Leur structure suit le standard OGR décrit en figure 11.1. Il est donc nécessaire de comprendre comment sont organisées les données à l'intérieur d'un fichier `shp`. L'exemple le plus simple que nous traiterons est celui du fichier `World_countries_shp.shp`

Le programme suivant, qui fait appel à la bibliothèque `osgeo.ogr` permet de récupérer des informations sur ce fichier.

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 bm_6.py
5
6 Purpose : Know what is inside a shapefile.
```

1. OGR signifiait à l'origine OpenGIS Simple Features Reference Implementation. Toutefois, OGR n'est pas totalement conforme à la spécification OpenGIS Simple Feature et n'est plus approuvée officiellement. La seule signification de OGR dans ce nom est historique.

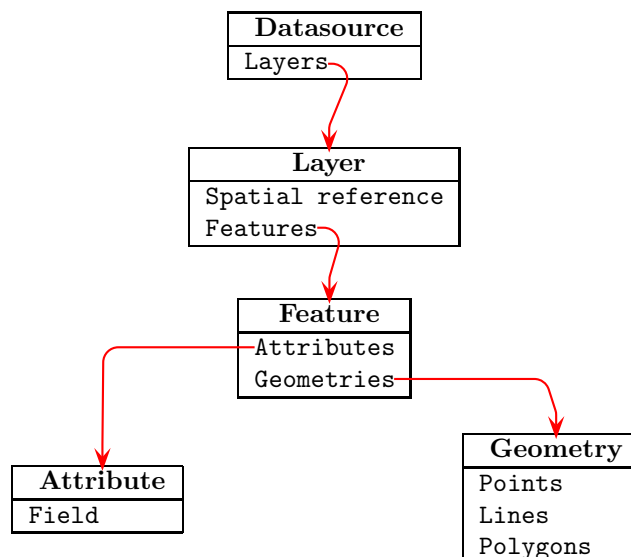


FIGURE 11.1 – Modèle de données OGR

```

7  """
8  import osgeo.ogr
9
10 # open shapefile
11 shapefile = osgeo.ogr.Open("../include/shapefiles/ne_50m_admin_0_countries/
    ne_50m_admin_0_countries.shp")
12
13 numLayers = shapefile.GetLayerCount() # counts layers
14 print( "Shapefile contains %d layers" % (numLayers))
15
16 # Gets the reference system and the number of features for each layer
17 # In the case of GEOFLA Each layer has a spatial reference ! this complicates
    everything !!!
18 for LayerNum in range(numLayers):
19     Layer = shapefile.GetLayer(LayerNum)
20     numFeatures = Layer.GetFeatureCount()
21     print( "Layer %d has %d features" % (LayerNum, numFeatures))
22     # tests whether there is a spatial ref attached to the file
23     if Layer.GetSpatialRef():
24         spatialRef = Layer.GetSpatialRef().ExportToProj4()
25         print( "Layer %d has spatial reference %s" % (LayerNum, spatialRef))
26     else:
27         print( "Layer %d has no spatial reference" % (LayerNum))
28
29     # gets the number of attributes per feature
30     for featureNum in range(numFeatures):
31         feature = Layer.GetFeature(featureNum)
32         fieldcount = feature.GetFieldCount()
33         print( "Feature %d has %d attributes\n" % (featureNum, fieldcount))
34
35     # prints the attributes of the last feature
36     print( "Feature %d has the following attributes:" % (featureNum))
37     attributes = feature.items()
38     for key, value in attributes.items():
39         print( "%s = %s" % (key, value))
40
41     # retrieve the feature geometry and print its type
42     geometry = feature.GetGeometryRef()
43     # geometries are stored in map units not in lat/lon this also complicates things
        for basemap.
44     print(geometry.Centroid().ExportToWkt())
  
```

```

45     geometryName = geometry.GetGeometryName()
46
47 print("Feature %d has a geometry of type %s" % (featureNum, geometryName))

```

Ce fichier à une couche (layer). Cette couche contient 238 objets. Chaque objet correspond à un pays (avec de erreurs cependant que vous pouvez chercher) et comporte un certain nombre d'attributs ainsi qu'une géométrie (ses frontières). un extrait de ce que renvoie le script précédent ressemble ainsi à

Feature 240 has the following attributes:

```

NAME_LEN =12
NAME_FR =Saint-Martin
ISO_A3 =SXM
ISO_A2 =SX
ABBREV_LEN =6
FIPS_10_ =NT
ECONOMY =6. Developing region
ADMO_A3_US =SXM
...

```

Commentaires.

1. Un défaut d'ogr : il ne reconnaît que les extensions écrites en minuscules (i.e .shp et non .SHP).
2. Les fichiers shapes peuvent être plus complexes et notamment contenir des données déjà projetées à la différence du fichier des pays qui ne contient que des données sous forme de lon/lat.
3. Même si vous ne référencez que le fichier .shp dans vos programmes un shapefile ne vient jamais seul. Trois fichiers forment un ensemble et ces trois fichiers sont nécessaires pour que vous puissiez utiliser le shapefile. le fichier shape (.shp) un fichier .shx et un fichier.dbf. Le shapefile contient les liens vers les deux autres fichiers et les géométries. le fichier .dbf contient les attributs dans une base de données au format DBF, le fichier .shx contient lui les index des données. D'autres fichiers peuvent être fournis en plus mais ces trois là sont obligatoires. Changez (temporairement !) l'extension du fichier .dbf pour voire.

11.4 Représenter un shapefile

11.4.1 Contours

cartopy

cartopy dispose d'un lecteur de shapefile qui à la façon de osgeo permet de lire et de récupérer les attributs et les géométries enregistrées dans le(s) fichier(s). cartopy implémente ensuite une méthode `add_geometries()` qui permet de représenter les géométries à la façon de polygones et de patch matplotlib

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  cm_7.py
5
6  Read a shapefile with cartopy
7  """
8  import cartopy.io.shapereader as shpreader
9  import cartopy.crs as ccrs
10 import matplotlib.pyplot as plt
11
12
13 reader = shpreader.Reader('../include/shapefiles/ne_50m_admin_0_countries/
    ne_50m_admin_0_countries.shp')
14 features= reader.records()
15
16 my_crs = ccrs.PlateCarree()
17
18 fig = plt.figure(figsize=(15/2.54,10/2.54))
19 ax = fig.add_subplot(111, projection = my_crs)
20
21
22 poly = []

```

```

23 # print the dictionary
24 for feature in features:
25     print('=====')
26     for key, val in feature.attributes.items():
27         print(key, val)
28
29 #print the last geometry
30 print(feature.geometry)
31
32
33 #plot the geometries
34 ax.add_geometries(reader.geometries(), crs=my_crs, facecolor='gray')
35 ax.coastlines()
36 ax.gridlines(draw_labels=True,xlocs=range(-180,181,60), ylocs=range(-90,91,30))
37
38 plt.savefig('../figures/cm_7.pdf',bbox_inches='tight')
39 plt.show()

```

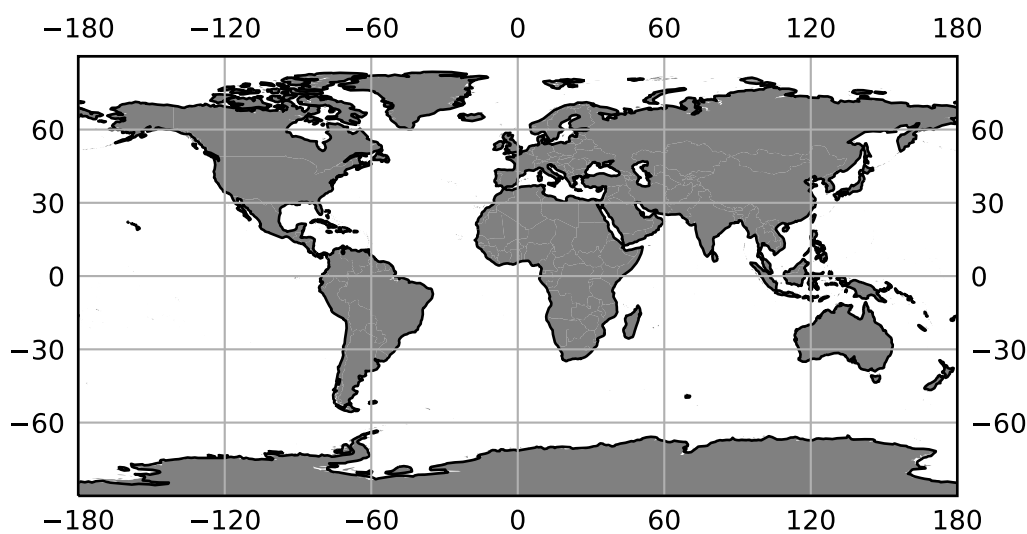


FIGURE 11.2 – Pays du monde, projection Plate Carrée

11.4.2 Colorier les polygones

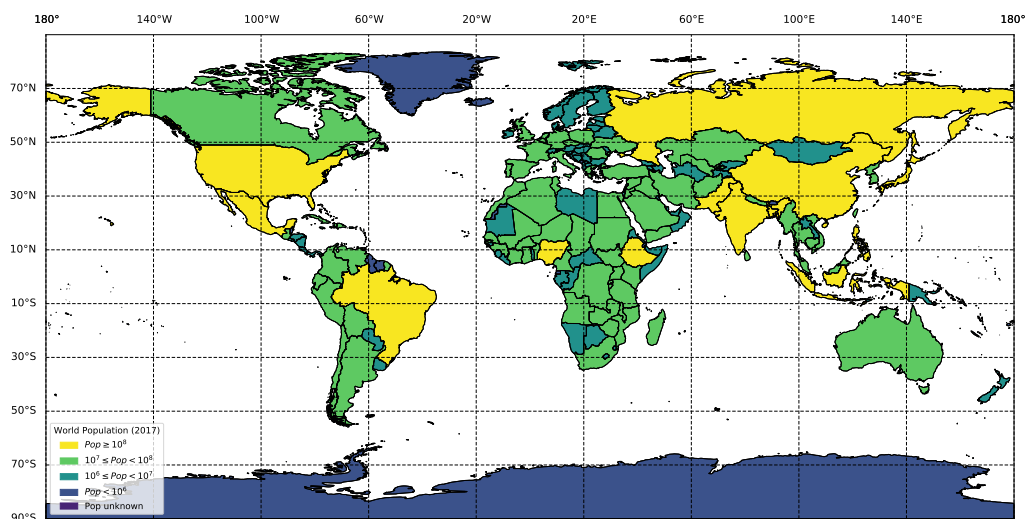


FIGURE 11.3 – Population mondiale (en 2017).

la librairie prévoit une méthode `add_geometries` directement dans `axis`. Les géométries sont des collections d'objets auxquelles on peut associer des propriétés de coloriage assez simplement comme le montre le script revu avec `cartopy`.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  cm_8.py
5
6  Fill polygons with colours
7  Note the ability of cartopy to reproject datasets
8  """
9
10 import cartopy
11 import cartopy.crs as ccrs
12 import cartopy.io.shapereader as shpreader
13
14 import matplotlib.pyplot as plt
15 from matplotlib.patches import Patch
16 from matplotlib import cm
17
18
19 # definition of the figure and axis
20 proj_crs = ccrs.Mollweide()
21 my_crs = ccrs.PlateCarree()
22 fig      = plt.figure(figsize = (20, 10))
23 ax       = fig.add_subplot(111, projection = proj_crs)
24
25 #color map I use viridis because it is nice !
26 cmap = cm.get_cmap('viridis')
27
28 reader = shpreader.Reader('../include/shapefiles/ne_50m_admin_0_countries/
29     ne_50m_admin_0_countries.shp')
30 features = reader.records()
31
32 #~ ax.gridlines(draw_labels=True)
33 ax.gridlines()
34 ax.coastlines()
35
36 for feature in features:
37     if feature.attributes['POP_EST'] >= 1e8:
38         ax.add_geometries([feature.geometry], facecolor = cmap(0.99), crs=
39             my_crs)
40     elif feature.attributes['POP_EST'] >= 1e7 and feature.attributes['POP_EST']
41         < 1e8:
42         ax.add_geometries([feature.geometry], facecolor = cmap(0.75),
43             edgecolor = 'k', crs=my_crs)
44     elif feature.attributes['POP_EST'] >= 1e6 and feature.attributes['POP_EST']
45         < 1e7:
46         ax.add_geometries([feature.geometry], facecolor = cmap(0.5),
47             edgecolor = 'k', crs=my_crs)
48     elif feature.attributes['POP_EST'] >= 0 and feature.attributes['POP_EST'] <
49         1e6:
50         ax.add_geometries([feature.geometry], facecolor = cmap(0.25),
51             edgecolor = 'k', crs=my_crs)
52     else:
53         ax.add_geometries([feature.geometry], facecolor = cmap(0.1),
54             edgecolor = 'k', crs=my_crs)
55
56 #and the legend
57 colors = [cmap(0.99), cmap(0.75), cmap(0.5), cmap(0.25), cmap(0.1)]

```



```

55 texts = ["$Pop_\geq 10^8$", "$10^7_\leq Pop_< 10^8$", "$10^6_\leq Pop_< 10^7$", "
           "$Pop_< 10^6$", "Pop_unknown"]
56 patches = [ Patch(color = colors[i], label = texts[i] ) for i in range(len(texts))
              ]
57 plt.legend(handles = patches, loc = 'lower_left', title = 'World_Population_(2017)'
             )
58
59
60 plt.show()

```

11.5 Les shapefiles comportant une projection

`cartopy` permet le changement de projection assez simplement. Il suffit de connaître et d'indiquer le système de projection dans lequel se trouve les données et de le passer dans l'argument `transform` ou `crs` afin que `cartopy` effectue de lui-même les transformations d'une projection vers l'autre.

Le plus simple pour le connaître est d'aller consulter le fichier `.prj` qui se trouve dans le même répertoire que le fichier `.shp`. Les informations de projection s'y trouveront. Pour les départements on trouve

```

PROJCS["RGF93_Lambert_93",GEOGCS["GCS_RGF_1993",
DATUM["D_RGF_1993",SPHEROID["GRS_1980",6378137.0,298.257222101]],
PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],
PROJECTION["Lambert_Conformal_Conic"],
PARAMETER["False_Easting",700000.0],PARAMETER["False_Northing",6600000.0],
PARAMETER["Central_Meridian",3.0],PARAMETER["Standard_Parallel_1",44.0],
PARAMETER["Standard_Parallel_2",49.0],PARAMETER["Latitude_Of_Origin",46.5],
UNIT["Meter",1.0]]

```

Après une rapide recherche (sur google ou `qwant` suivant votre degré d'addiction) on trouve que la projection RGF93/Lambert 93 a un code EPSG de 2154. la suite se trouve dans le script et c'est indéniablement plus simple...

```

1  """
2  Reprojecter les données d'un shapefile avec cartopy
3  """
4  import cartopy.crs as ccrs
5  import cartopy.io.shapereader as shapereader
6
7  import matplotlib.pyplot as plt
8  import numpy as np
9
10 reader = shapereader.Reader("./shapefiles/DEPARTEMENTS/DEPARTEMENT.SHP")
11
12 fig = plt.figure()
13 ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
14 ax.coastlines(resolution='10m')
15 ax.gridlines(draw_labels=True)
16 ax.set_xlim(-5,10)
17 ax.set_ylim(40,55)
18
19 # epsg code replaces the complete projection definition. it can be found easily on
   the web
20 ax.add_geometries(reader.geometries(), crs=ccrs.epsg(2154))
21
22
23 plt.show()

```

11.6 Écrire des données dans un shapefile

Imaginons que nous voulons enregistrer un jeu de données dans un `shapefile`. Nous voulons enregistrer la localisation de l'IPGP (48.844855,2.356685) et de l'UFR STEP (48.827646,2.380656) dans notre fichier.

La procédure est alors la suivante (script `create-shapefile.py`) :

1. créer le `shapefile` définition et sa référence spatiale éventuelle,

2. créer un attribut (champs) correspondant au nom de lieu,
3. créer la géométrie correspondante,
4. créer l'objet et le remplir de son attribut et de sa géométrie.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  bm_10.py
5
6  Write data to a shapefile
7  """
8  #import libraries
9  from osgeo import ogr, osr
10 import sys, os
11
12 # set the spatial reference
13 spatialReference = osr.SpatialReference()
14 spatialReference.SetWellKnownGeogCS('WGS84')
15
16 # create the shapefile
17 driver = ogr.GetDriverByName('ESRI_Shapefile')
18 dstFile = driver.CreateDataSource('../include/shapefiles/IPGP/IPGinfo.shp')
19 if dstFile is None:
20     print("could not create file")
21     sys.exit(1)
22
23 # create the layer with the spatial reference defined above
24 layer = dstFile.CreateLayer("IPGInfo", spatialReference)
25
26 # create the field where we will store the name of places
27 fieldDef = ogr.FieldDefn("Name", ogr.OFTString)
28 fieldDef.SetWidth(50)
29 layer.CreateField(fieldDef) # add the field definition to the layer
30
31 """
32 create the two features
33 """
34 # the IPGP feature
35 featureDefn = layer.GetLayerDefn()
36 feature = ogr.Feature(featureDefn)
37 feature.SetField('Name', 'IPGP-Cuvier')
38
39 #create the feature's geometry
40 point = ogr.Geometry(ogr.wkbPoint)
41 point.AddPoint(48.844855, 2.356685)
42 feature.SetGeometry(point)
43
44 #add the feature to the layer
45 layer.CreateFeature(feature)
46 # destroy a feature when finished so it is saved
47 feature.Destroy()
48
49 #do it again for UFR
50 featureDefn = layer.GetLayerDefn()
51 feature = ogr.Feature(featureDefn)
52 feature.SetField('Name', 'IPGP-Lamarck')
53 point = ogr.Geometry(ogr.wkbPoint)
54 point.AddPoint(48.827646, 2.380656)
55 feature.SetGeometry(point)
56 layer.CreateFeature(feature)
57 feature.Destroy()
58 #same for the file a bit like close
59 dstFile.Destroy()

```

Une fois que c'est fait on peut vérifier que tout va bien en utilisant le script du §11.3

Chapitre 12

Les rasters

12.1 Représenter un raster

Les rasters sont des fichiers matriciels c'est à dire qu'ils contiennent une ou plusieurs matrices de nombres. Une cas particulier très courant correspond aux fichiers images. Oui mais dans notre cas :

- les pixels correspondent à une surface d'une certaine largeur/longueur sur le globe;
- comme tout objet plan ils correspondent à une projection d'un objet réel (pris par exemple par les capteurs d'un instrument);
- afin de les représenter correctement on doit donc connaître leur projection, l'origine de l'image (souvent le coin en haut à gauche) et la taille des pixels.

12.1.1 Landsat prétraitée de l'Aquitaine

Suivant son format un raster viendra seul ... ou pas. S'il est accompagné de fichiers de métadonnées qui donnent les informations spatiales conservez tous les fichiers dans le même dossier et ne jetez pas ces fichiers! Exemple avec la belle région d'Aquitaine en particulier les Landes. L'image est un fichier jpeg, elle ne contient donc aucune information géographique, celles-ci sont enregistrées dans des fichiers comme un .xml. Vous n'aurez pas à vous en préoccuper, la seule chose à faire c'est de ne pas y toucher.

On peut alors se servir des informations pour représenter notre belle image avec en prime la belle ville de Bordeaux (figure 12.1).

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 cm_12.py
5
6 Plot a georeferenced jpeg image
7 Add a point location and its name
8 we use gdal to get the image info and
9 cartopy to plot
10 """
11
12 import cartopy.crs as ccrs
13 import matplotlib.pyplot as plt
14 from osgeo import gdal
15 from osgeo import osr
16 import pyproj
17
18 import numpy as np
19
20
21 ds = gdal.Open("../include/raster/Aquitaine/LC82000292015341LGN00.jpg")
22 data = ds.ReadAsArray()
23
24 # map limits
25
26 geotransform = ds.GetGeoTransform()
27
28 #upper left corner
29 originX = geotransform[0]
```

```

30 originY = geotransform[3]
31
32 print( originX,  originY )
33
34 #image width and height
35 pixelWidth = geotransform[1]
36 pixelHeight = geotransform[5]
37 print( pixelWidth,  pixelHeight )
38 W = abs(pixelWidth * ds.RasterXSize)
39 H = abs(pixelHeight * ds.RasterYSize)
40
41 # get the lower right corner
42 CornerX= originX + W
43 CornerY= originY - H
44
45 #Image extent
46 extent = [originX, CornerX, CornerY, originY]
47
48
49
50 #figure
51 fig = plt.figure(figsize = (20, 20)) # because it is a raster the output size is
    important
52 ax=fig.add_subplot(111, projection=ccrs.UTM(30))
53
54 ax.imshow(data[:3,:,:].transpose((1,2,0)), origin = 'upper', extent=extent,
    transform=ccrs.UTM(30))
55
56 #place Bordeaux to test
57 (x, y) = (-0.568058, 44.8409391)
58 ax.plot(x, y, 'ro', ms = 12, transform=ccrs.Geodetic())
59 ax.text(1.01*x, 1.01*y, 'Bordeaux', fontsize = 12, color = 'w')
60
61 ax.plot(originX,originY, 'bo', ms = 12, transform=ccrs.UTM(30))
62 ax.plot(CornerX,CornerY, 'bo', ms = 12, transform=ccrs.UTM(30))
63 ax.coastlines(resolution='10m')
64 ax.gridlines()
65
66 plt.show()

```

12.1.2 Topographie de la Guadeloupe

Ce dont nous disposons comme informations :

- la topographie est établie à partir des images du satellite ASTER de résolution de 90m ;
- le format de fichiers est du **Geotiff** = fichier raster contenant des metadonnées sur la position dans l'espace et la projection de l'image ;
- les fichiers ASTER GDEM sont en lon/lat sans projection particulière ;
- les fichiers fournis en tuiles de 1° de côté.
- Pour la Guadeloupe (et les antilles) on en a deux.

Procédure :

- on utilise la bibliothèque GDAL pour ouvrir le fichier
- on extrait les coordonnées des coins
- **cartopy**
 - on définit la projection de la figure
 - on représente les deux MNT l'un à la suite de l'autre
 - on fait le travail de finition (côtes, grille etc...)

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 cm_12-2.py
5
6
7 Plot a georeferenced DEM
8 using merging and cropping

```

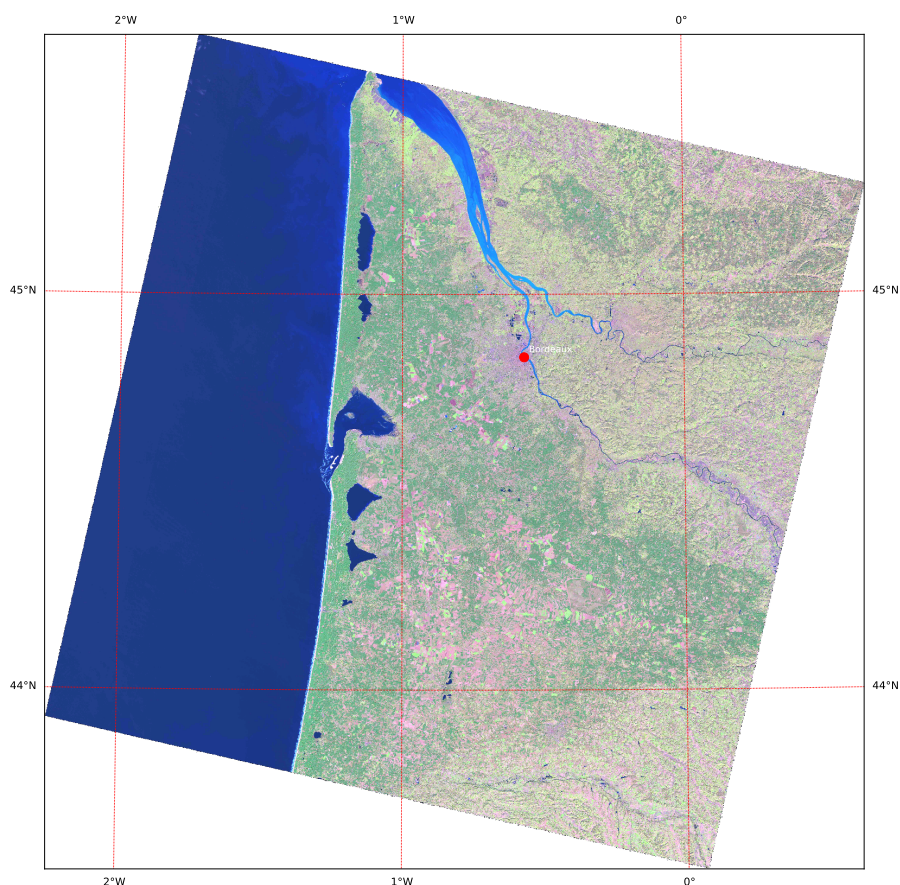


FIGURE 12.1 – Landsat TM8 des landes d'Aquitaine.

```

9
10 Cartopy version
11 """
12
13 import cartopy.crs as ccrs
14 import matplotlib.pyplot as plt
15 from osgeo import gdal
16 from osgeo import osr
17 import pyproj
18 from numpy import linspace
19 from numpy import meshgrid
20
21 import sys
22 sys.path.append('/usr/bin/')
23 import gdal_merge as gm
24
25
26 fname = ["../include/raster/Guadeloupe/ASTGTM2_N16W062/ASTGTM2_N16W062_dem.tif",\
27          "../include/raster/Guadeloupe/ASTGTM2_N15W062/ASTGTM2_N15W062_dem.tif"]
28
29 fig = plt.figure(figsize = (20, 20))# because it is a raster the output size is
      important
30 ax = fig.add_subplot(111, projection = ccrs.PlateCarree())
31
32 minx=180
33 maxx=-180
34 miny=90
35 maxy=-90
36
37 for f in fname:
38
39     ds = gdal.Open(f)

```

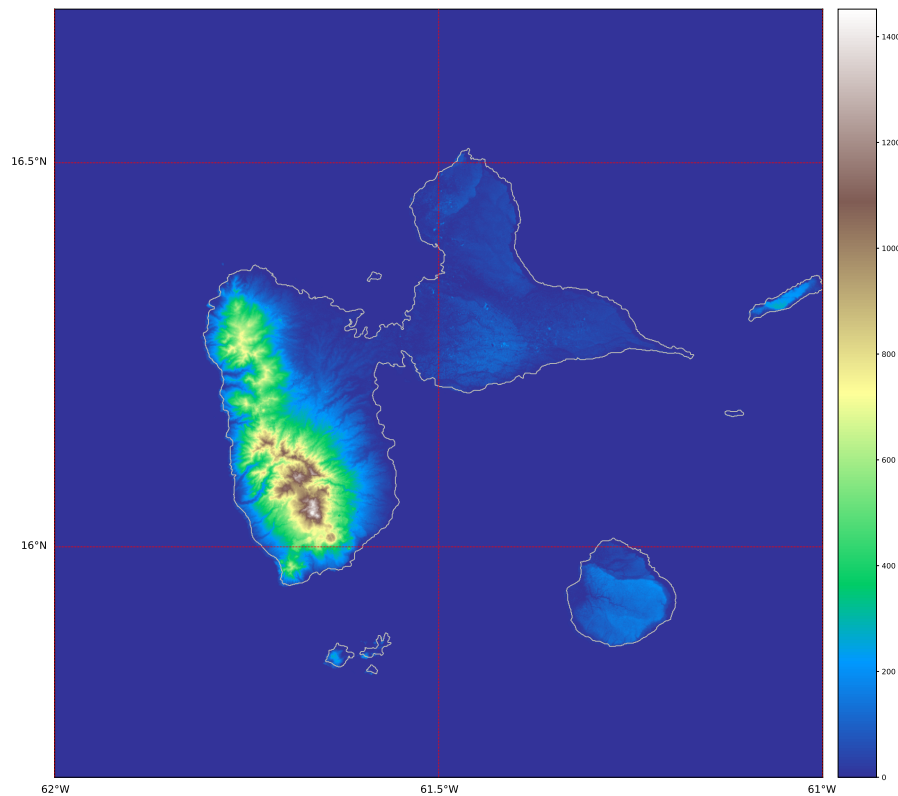


FIGURE 12.2 – MNT ASTER de la Guadeloupe...en un seul bout! CHANGER OUR LE BON MNT GUAD + MARTI

```

40 data = ds.ReadAsArray()
41
42
43 # map limits
44 geotransform = ds.GetGeoTransform()
45 proj=ds.GetProjection()
46 print(proj)
47 originX = geotransform[0]
48 originY = geotransform[3]
49 print( originX, originY )
50 pixelWidth = geotransform[1]
51 pixelHeight = geotransform[5]
52 print( pixelWidth, pixelHeight )
53
54 w = abs(pixelWidth * ds.RasterXSize)
55 h = abs(pixelHeight * ds.RasterYSize)
56
57 #corners
58 lx = originX
59 ly = originY-h
60 rx = lx+w
61 ry = originY
62
63 if minx > min(lx,rx):
64     minx = min(lx,rx)
65 if maxx < max(lx,rx):
66     maxx = max(lx,rx)
67 if miny > min(ly,ry):
68     miny = min(ly,ry)
69 if maxy < max(ly,ry):
70     maxy = max(ry,ly)
71
72 extent = (originX, originX + abs(pixelWidth * ds.RasterXSize) , originY -
           abs(pixelHeight * ds.RasterYSize) , originY)

```

```

73     print(extent)
74
75     print('lower_left_corner', lx, ly )
76     print('upper_right_corner', rx, ry )
77
78     #get the spatial projection of the DEM
79
80     inproj = osr.SpatialReference()
81     inproj.ImportFromWkt(proj)
82     projcs = inproj.GetAuthorityCode('PROJCS')
83     print(projcs)
84     #~ DEM_proj = ccrs.epsg(6326)
85
86
87     ax.imshow(data, extent=extent, origin='upper', cmap='terrain', transform=ccrs
88             .PlateCarree())
89
90 ax.coastlines(resolution='10m')
91 ax.gridlines(draw_labels=True)
92 ax.set_xlim(minx,maxx)
93 ax.set_ylim(miny,maxy)
94 # plt.savefig("../figures/antillespdf", bbox_inches = "tight")
95 plt.show()

```

12.2 Les images GeoTiff

C'est le format le plus souvent fournit par les fournisseurs d'images comme les images Landsat ou Spot par exemple. Ce format inclut un entête avec toutes les informations nécessaire à la représentation spatiale. Conséquence : n'essayez pas de bidouiller les couleurs avec Gimp ou Photoshop. Vous risquez de perdre ces précieuses informations.

Nous allons travailler sur une image Landsat d'une région interne de la chaîne de montagne du Tianshan en Chine, le bassin de Bayanbulak, et utiliser GDAL un peu mieux. Nous disposons de trois images correspondant à trois longueurs d'ondes. La première chose que nous pouvons faire c'est d'appliquer nos connaissances précédentes pour voir à quoi ressemble une des images, la L5144030_03020060731_B10.TIF par exemple.

L'application des scripts est assez simple —je vous la laisse en exercice—. On obtient la figure 12.3

Passé la minute de fierté on se rend compte qu'il reste encore un peu de travail avant d'obtenir quelque chose de satisfaisant. et là je le répète hors de question de jouer avec Gimp ou Photoshop pour vous en sortir. Heureusement il y a `gdal` !

La procédure sera la suivante :

1. récupérer pour chacune de nos trois images l'histogramme des valeurs de pixels ;
2. étirer un peu tout cela ;
3. fondre les images en une seule à trois canaux pour obtenir une image en couleur.

12.2.1 Histogramme

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  bm_13.py
5
6  Open Raster bands from a Geotiff Landsat 5 image
7  Extract and plot the histograms
8  """
9
10 from osgeo import gdal, osr, ogr
11 from osgeo.gdalconst import *
12 import numpy as np
13 import matplotlib.pyplot as pl
14 import pyproj
15
16 # get the GeoTiff driver and register it

```

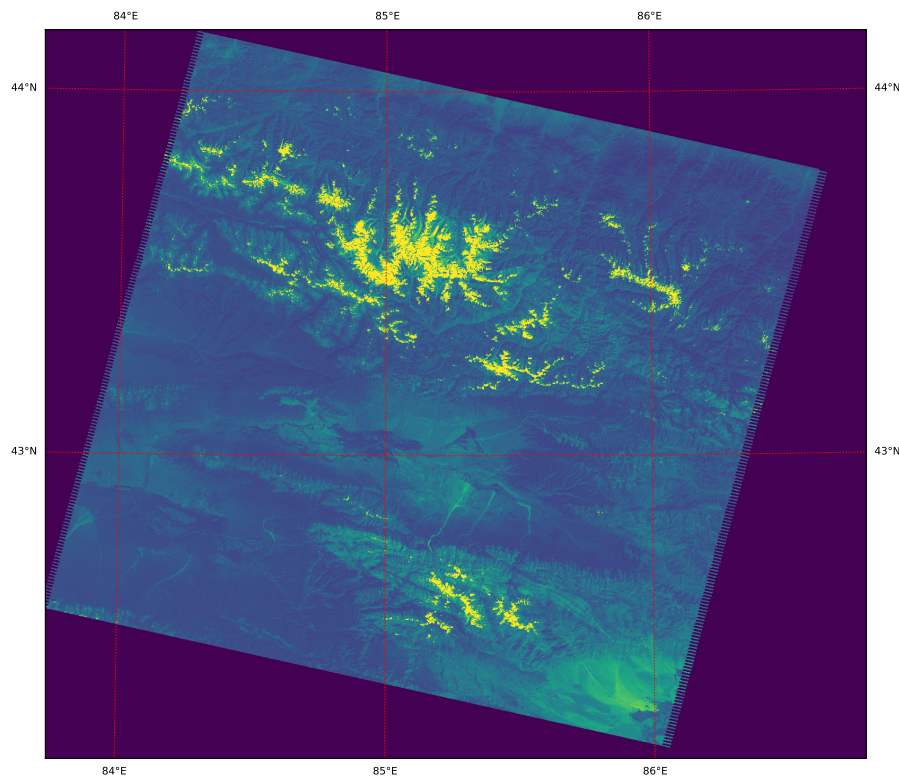



FIGURE 12.3 – Bassin de Bayanbulak Landsat 5 144030, Canal 10.

```

17 driver = gdal.GetDriverByName('GTiff')
18 driver.Register()
19
20
21 def landsat_hist(fn):
22
23     pl.figure(figsize = (8, 11))
24     bcount = 1
25     for f in fn:
26         ds = gdal.Open(f, GA_ReadOnly)
27         if ds is None:
28             print( 'Could not open' + f )
29             sys.exit(1)
30
31         #raster size
32         cols = ds.RasterXSize
33         rows = ds.RasterYSize
34         bands = ds.RasterCount
35
36         print( bands, rows, cols )
37
38         # Transform informations
39         geotransform = ds.GetGeoTransform()
40         originX = geotransform[0]
41         originY = geotransform[3]
42         pixelWidth = geotransform[1]
43         pixelHeight = geotransform[5]
44
45         print( originX, originY, pixelWidth, pixelHeight )
46
47         #read band and compute histogram
48         band = ds.GetRasterBand(1)
49         data = band.ReadAsArray(0, 0, cols, rows)
50         h, b = np.histogram(data, 255)
51         pl.subplot(3, 1, bcount)
52         pl.semilogy(b[2:250], h[2:250], 'r-')

```

```

53     pl.ylabel('N')
54     pl.title(f)
55     bcount = bcount+1
56     pl.xlabel('pixel value')
57
58     pl.show()
59
60
61
62 fn = ["../include/raster/China/144030/L5144030_03020060731_B10.TIF", \
63       "../include/raster/China/144030/L5144030_03020060731_B20.TIF", \
64       "../include/raster/China/144030/L5144030_03020060731_B30.TIF"]
65 landsat_hist(fn)

```

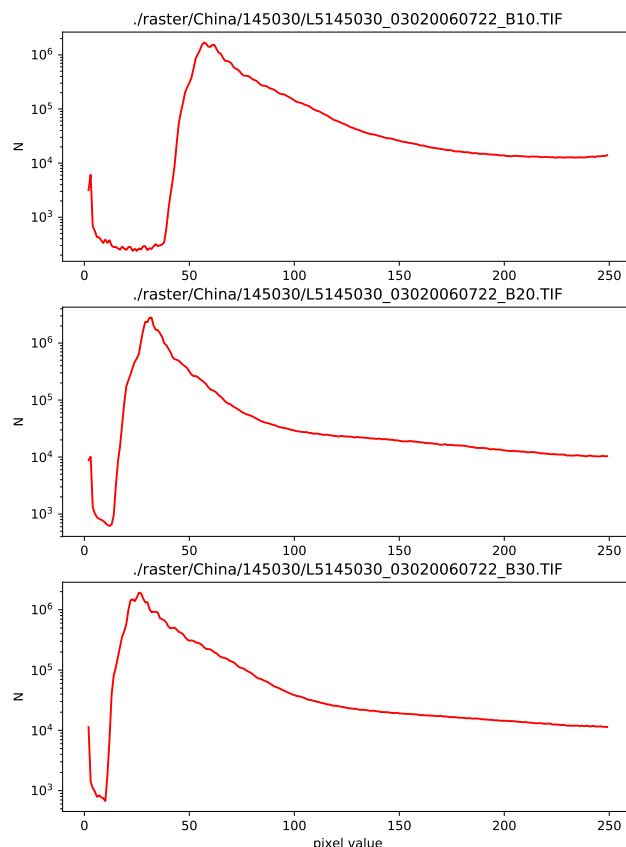


FIGURE 12.4 – Histogramme des trois canaux (10,20,30) de la Landsat 144030

12.2.2 Étirement des valeurs de pixels et fusion des canaux

L'étirement passe par la fonction `gdal.Translate()`. Une fois que vous avez choisi vos limites le script pour chaque image est simple

```

1 data=gdal.Open("../raster/China/144030/L5144030_03020060731_B10.TIF")
2 gdal.Translate(destName="../raster/China/144030/B10_s.TIF", srcDS=data, scaleParams
  =[[30, 200, 1, 254]], ,exponents=[0.7])

```

les deux arguments de transformation de l'histogramme sont

1. `scaleParams` : définit les valeurs min max de départ et d'arrivée
2. `exponents` : définit la déformation de l'histogramme. 1= non déformé ; entre 0 et 1 décale le pic vers les valeurs les plus claires ; au dessus de 1 décale le pic vers les valeurs les plus foncées.

Une fois que l'on a fait ce travail pour les trois canaux on peut alors fusionner ceux-ci dans une seule image. Pour y arriver nous devons utiliser un script python appelé `gdal_merge.py` fourni avec la librairie `gdal` mais non intégré à celle-ci (ne me demandez pas pourquoi). Normalement cette fonction est située dans `/usr/bin`. le script donne ceci

```

1 import sys
2 sys.path.append('/usr/bin/')
3 import gdal_merge as gm
4
5 gm.main(['', '-o', './raster/China/144030/144030_rgb.tif', \
6 '-separate', './raster/China/144030/B30_s.TIF', \
7 './raster/China/144030/B20_s.TIF', './raster/China/144030/B10_s.TIF'])

```

Commentaires :

1. l'appel de `gdal_merge.py` est un peu lourd car il faut appeler la fonction et lui transmettre tous les arguments comme une liste de chaînes de caractères...
2. Si vous tentez de refaire tourner le programme il faudra effacer l'image créée antérieurement.

Notre Landsat ressemble maintenant à la figure 12.5. C'est nettement mieux n'est-il-pas? On pourrait s'en contenter mais non...

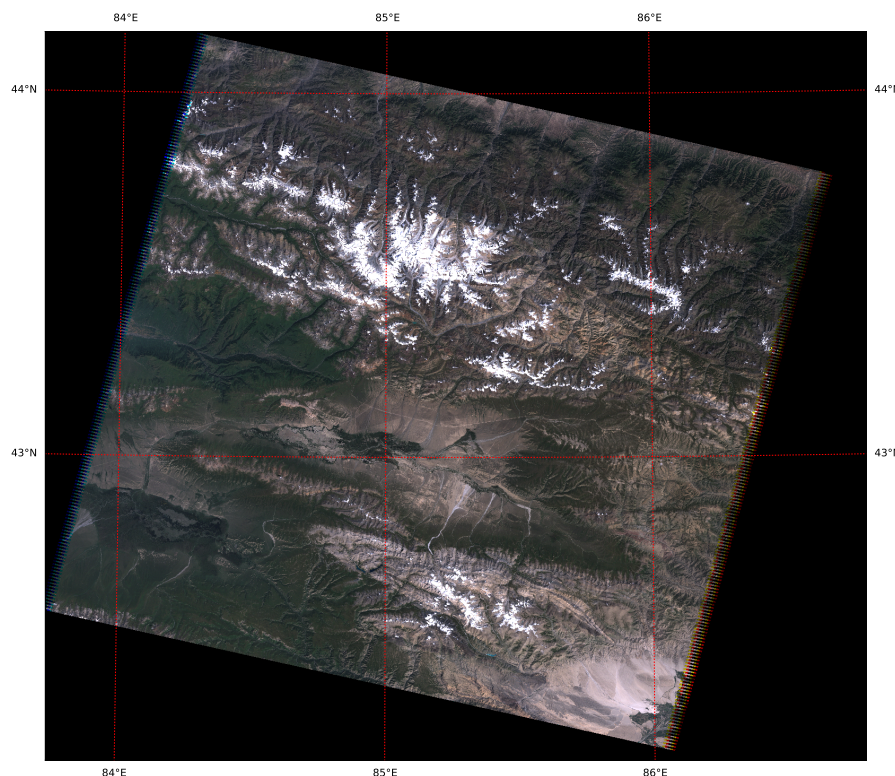


FIGURE 12.5 – Bassin de Bayanbulak, Landsat 5 144030; image composée des trois canaux (R :30,V :20,B :10) étirés

12.2.3 Joindre et découper

Le bassin de Bayanbulak correspond en fait aux deux bassins que l'on voit sur la figure 12.5. Pour l'avoir dans son entier il faut fusionner deux images Landsat. La figure 12.6 montre la seconde image un peu plus à l'est de la première.

Afin de joindre les deux images, il faut (1) les projeter dans le même système de coordonnées et (2) les découper afin de les coller avec une légère superposition (on pourrait les superposer complètement mais vous aurez remarqué que les bords d'une image Landsat projetée sont crénelés et si on ne recoupe pas et bien cela se verra).

Pour nous simplifier la vie, qui plus est ce sera utile plus tard, nous allons projeter tout le monde en Plate Carrée (long/lat) puis nous couperons et nous collerons. Si vous avez réussi à produire les figures 12.5 et 12.6, la suite prend trois lignes de code...

```

1 gdal.Warp("./raster/China/144030/144030_rgb_ll.tif", \
2           "./raster/China/144030/144030_rgb.tif", dstSRS="+proj=latlong+datum=WGS84", \
3           outputBounds=(84.25, 42.5, 86.1, 43.5))
4 gdal.Warp("./raster/China/145030/145030_rgb_ll.tif", \
5           "./raster/China/145030/145030_rgb.tif", dstSRS="+proj=latlong+datum=WGS84", \

```

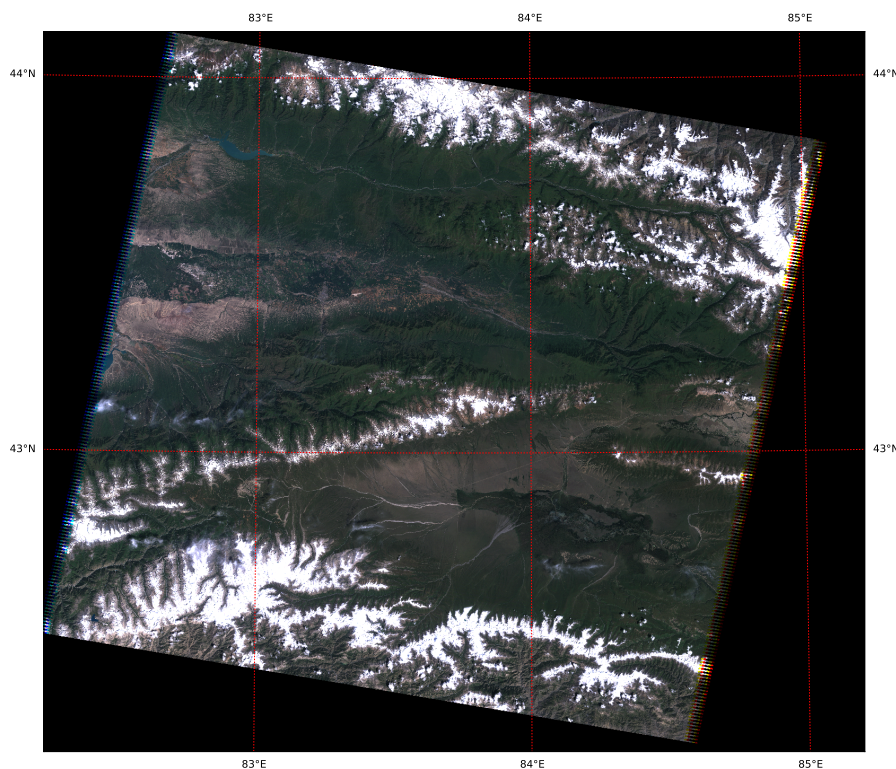


FIGURE 12.6 – Bassin de Bayanbulak Landsat 5 145030, image composée des trois canaux (R :30,V :20,B :10) étirés

```

6         outputBounds=(82.5, 42.5, 84.5, 43.5)
7
8 gm.main(['', '-n', '0', '-o', './raster/China/bayan.tif', \
9 './raster/China/144030/144030_rgb_ll.tif', \
10 './raster/China/145030/145030_rgb_ll.tif'])

```

Ne reste plus qu'à créer la carte

Le script suivant résume l'ensemble des opérations :

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 bm_14.py
5
6 (i) Stretch the images according to the histograms obtained using bm_13
7 (ii) Merge the bands in one multi-layered raster band
8 (iii) Reproject and crop the raster
9 (iv) merge to obtain the final image
10 (v) plot the map.
11
12 """
13
14
15 import sys
16 sys.path.append('/usr/bin/')
17
18
19 from osgeo import gdal
20 import gdal_merge as gm
21 import matplotlib.pyplot as plt
22 import cartopy.crs as ccrs
23
24 # get the GeoTiff driver and register it
25 driver = gdal.GetDriverByName('GTiff')
26 driver.Register()
27

```

```

28 def process_geotiffs():
29     # (i)
30     # Stretch the histograms
31     # 144030
32
33     data = gdal.Open("../include/raster/China/144030/L5144030_03020060731_B10.TIF")
34     gdal.Translate(destName = "../include/raster/China/144030/B10_s.TIF", srcDS =
35         data, scaleParams = [[30, 200, 1, 254]], exponents = [0.7])
36     data = gdal.Open("../include/raster/China/144030/L5144030_03020060731_B20.TIF")
37     gdal.Translate("../include/raster/China/144030/B20_s.TIF", data, scaleParams =
38         [[10, 120, 1, 254]], exponents = [0.7])
39     data = gdal.Open("../include/raster/China/144030/L5144030_03020060731_B30.TIF")
40     gdal.Translate("../include/raster/China/144030/B30_s.TIF", data, scaleParams =
41         [[8, 140, 1, 254]], exponents = [0.7])
42
43     # 145030
44     data = gdal.Open("../include/raster/China/145030/L5145030_03020060722_B10.TIF")
45     gdal.Translate(destName = "../include/raster/China/145030/B10_s.TIF", srcDS =
46         data, scaleParams = [[30, 200, 1, 254]], exponents = [0.7])
47     data = gdal.Open("../include/raster/China/145030/L5145030_03020060722_B20.TIF")
48     gdal.Translate("../include/raster/China/145030/B20_s.TIF", data, scaleParams =
49         [[10, 120, 1, 254]], exponents = [0.7])
50     data = gdal.Open("../include/raster/China/145030/L5145030_03020060722_B30.TIF")
51     gdal.Translate("../include/raster/China/145030/B30_s.TIF", data, scaleParams =
52         [[8, 140, 1, 254]], exponents = [0.7])
53
54     # (ii)
55     # Merge the bands
56     #
57     gm.main(['', '-o', '../include/raster/China/144030/144030_rgb.tif', '-separate',
58         '../include/raster/China/144030/B30_s.TIF', '../include/raster/China
59         /144030/B20_s.TIF', '../include/raster/China/144030/B10_s.TIF'])
60     gm.main(['', '-o', '../include/raster/China/145030/145030_rgb.tif', '-separate',
61         '../include/raster/China/145030/B30_s.TIF', '../include/raster/China
62         /145030/B20_s.TIF', '../include/raster/China/145030/B10_s.TIF'])
63
64     # (iii)
65     # Crop and reproject the images
66     #
67     gdal.Warp("../include/raster/China/144030/144030_rgb_ll.tif", "../include/raster/
68         China/144030/144030_rgb.tif", dstSRS = "+proj=latlong+datum=WGS84",
69         outputBounds = (84.25, 42.5, 86.1, 43.5)) # NB: no spaces surrounding '='
70         in the SRS !!!
71     gdal.Warp("../include/raster/China/145030/145030_rgb_ll.tif", "../include/raster/
72         China/145030/145030_rgb.tif", dstSRS = "+proj=latlong+datum=WGS84",
73         outputBounds = (82.5, 42.5, 84.5, 43.5)) # NB: no spaces surrounding '='
74         in the SRS !!!
75
76     #~ # (iv)
77     #~ # Merge into the final image
78     #~ #
79     gm.main(['', '-n', '0', '-o', '../include/raster/China/bayan.tif', '../include/
80         raster/China/144030/144030_rgb_ll.tif', '../include/raster/China
81         /145030/145030_rgb_ll.tif'])
82
83     # if needed
84     # process_geotiffs()
85
86     #
87     # Draw the map
88     #
89     ds = gdal.Open("../include/raster/China/bayan.tif")
90     data = ds.ReadAsArray()

```

```

76
77
78 fig = plt.figure(figsize = (20, 10))# because it is a raster the output size matters
79
80 # If you do not need a fancy projection use PlateCarree. It's much faster because
    there is no need to reproject the image pixels
81 ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
82
83 im_extent = [82.5, 86.1, 42.5, 43.5] # [left_x, right_x, bottom_y, top_y]
84 ax.imshow(data[:3,:,:].transpose((1,2,0)), extent = im_extent, origin = 'upper',
    transform=ccrs.PlateCarree())
85
86 # Using platecarree you can also plot your labels !
87 ax.gridlines(draw_labels=True, xlocs=range(82,86,1),ylocs=[42.5,43,43.5])
88
89 # Finally set the extent
90 ax.set_xlim(82.5,86.1)
91 ax.set_ylim(42.5,43.5)
92
93 plt.show()

```

Et là, avouons le, le résultat est satisfaisant (figure 12.7). Vous aurez compris qu'il est on ne peut plus simple de reprojeter l'image bayan.tif dans le système de coordonnées qui vous plaît ! La figure 12.7 représente d'ailleurs le même bassin mais cette fois la projection est azimutale équidistante. la commande pour passer de l'une à l'autre fera appel à `gdal.Warp`

```

1 gdal.Warp("./raster/China/bayan_aeqd.tif", "./raster/China/bayan.tif", dstSRS="+
    proj=aeqd_+lon_0=84_+lat_0=43_+datum=WGS84")

```

En outre, pour permettre une comparaison, nous n'avons pas utilisé le décalage de pic (exponents) de `gdal.Translate()`. L'image est donc plus sombre.

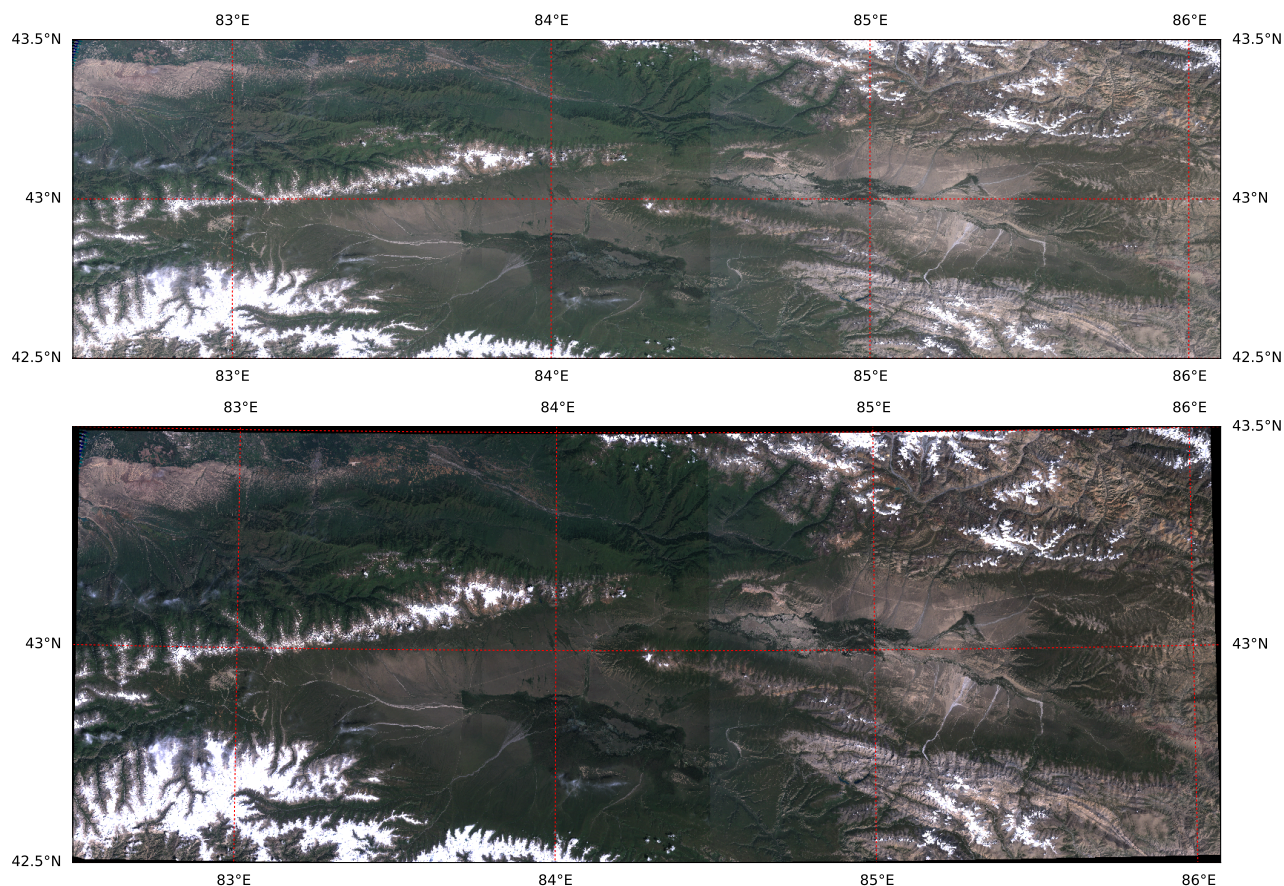


FIGURE 12.7 – Bassin de Bayanbulak. image composée à partir des trois canaux (R :30,V :20,B :10) étirés de deux images Landsat 5 (144030, 145030). Projections plate carrée et azimutale équidistante.

Chapitre 13

Exemples d'applications

Dans ce qui suit je présente différents exemples d'applications. Je donne les scripts. Les représentations graphiques sont à peu de choses près toujours les même. On notera toujours cette simplicité apportée par `cartopy` dans la gestion des projections, notamment la possibilité de reprojeter des données dans le système de la figure, et dans la gestion des collections, notamment pour la shapefiles. On notera aussi que pour les labels `cartopy` reste encore à améliorer.

13.1 Base des Parcours

Reprenons notre travail sur les bases de données et notamment sur Parcours. On voulait savoir où étaient les étudiants l'année précédent leur arrivée à l'UFR. Nous avons créé une vue `OneYearBefore` permettant de stocker la requête. Nous allons pouvoir utiliser nos connaissances de SQL et de python pour joindre ces données aux géométries des départements. L'objectif étant de colorier chaque département en fonction du nombre d'étudiants qui s'y trouvent.

Procédure

1. Connexion à la base de donnée `Parcours` et récupération des données au moyen d'une requête.
2. Création d'un dictionnaire et d'une liste pour établir la jointure avec les code des départements de la carte.
3. Initialisation de la figure et création de la carte de base à partir du fichier départements re-projeté précédemment.
4. Création de la table de couleurs connaissant le nombre max d'étudiants (64) dans un département.
5. Récupération des polygones et des infos.
6. Test sur les codes de département et appariement avec les données récupérées de la base.
7. Création des polygones `matplotlib` et remplissage.

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 cm_15.py
5
6 Map the French départements with the result of a query performed in
7 the MySQL Parcours database.
8
9
10 cartopy version
11 """
12
13 #libraries
14
15 import matplotlib.pyplot as plt
16 import matplotlib.cm as cm
17 import matplotlib as mpl
18
19 from matplotlib.patches import Patch
20 from mpl_toolkits.axes_grid1.inset_locator import inset_axes
21
22 import cartopy.crs as ccrs
```



```

23 import cartopy.io.shapereader as shpreader
24 import cartopy.feature as cfeature
25
26
27 import numpy as np
28
29 from sqlalchemy import create_engine
30
31
32 #CONNEXION
33 # name of the database you wish to connect to
34 base = 'Parcours'
35 # establish connection with sqlalchemy
36 # a simple way to store login info
37 f = open('./identifiers.txt')
38 mylogin = f.readline().strip('\n')
39 mypass = f.readline().strip('\n')
40
41 engine = create_engine('mysql://%s:%s@localhost/Parcours' % (mylogin,mypass))
42 conn = engine.connect()
43
44
45 query = """select dep, count(*) as N from
46         (select substr(cp,1,2) as dep
47          from ParcoursTbl p inner join IdentiteTbl i on p.idetudiant=i.idetudiant
48          where annee=anneeufr-1 and pays='France') as Q
49         group by dep ;
50         """
51 # execute the query
52 rows = conn.execute(query)
53
54 #DICTIONNARY AND LIST
55 # We are going to create a dictionnary with the number of
56 # students as the value and the departemnt code as the key
57 # and a list of keys
58 OYB = {}
59 keys = []
60 for r in rows:
61     OYB[r[0]] = float(r[1])
62     keys.append(r[0])
63
64
65 #FIGURE
66 fig = plt.figure()
67 ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
68
69 #set map limits
70 ax.set_xlim(-5,10)
71 ax.set_ylim(40,52)
72
73 # the first time it may take time to load into cache
74 ocean_50m = cfeature.NaturalEarthFeature('physical', 'ocean', '50m',
75                                         edgecolor='face',
76                                         facecolor=cfeature.COLORS['water'])
77
78 ax.add_feature(ocean_50m)
79
80 #you have to go beyhond the limits if you want the grid to extend until the limits.
81 #remind that up to now only PlateCarree offers the possibility to draw labels.
82 ax.gridlines(draw_labels=True,xlocs=range(-5,12,2), ylocs=range(40,54,2), zorder=4)
83
84 #get the department
85 reader= shpreader.Reader("../include/shapefiles/DEPARTEMENTS/DEPARTEMENT_LATLON.shp")
86 features = reader.records()
87
88 #loop through departments

```

```

89 #COLORMAP
90 #selects and colors the departements
91 cmap = cm.get_cmap('viridis')
92 c = (0.9, 0.9, 0.9)
93 for feature in features:
94     dep = feature.attributes['CODE_DEPT']
95     if dep in keys:
96         if OYB[dep]>50:
97             c = cmap(0.99)
98         elif OYB[dep] >= 10 and OYB[dep] < 50:
99             c = cmap(0.75)
100        elif OYB[dep] >= 5 and OYB[dep] < 10:
101            c = cmap(0.5)
102        elif OYB[dep] >= 1 < 5:
103            c = cmap(0.25)
104        else:
105            c = (0.9, 0.9, 0.9)
106
107        ax.add_geometries([feature.geometry], edgecolor = 'k', \
108            linewidth = 0.5, facecolor = c, zorder = 2, crs=ccrs.PlateCarree())
109    else:
110        ax.add_geometries([feature.geometry], edgecolor = 'k', \
111            linewidth = 0.5, facecolor = 'white', zorder = 2, crs=ccrs.
112                PlateCarree())
113
114 #and the legend
115 colors = [cmap(0.99), cmap(0.75), cmap(0.5), cmap(0.25), (0.9, 0.9, 0.9)]
116 texts = ["$N_\u226450$", "$10_\u2264N_\u226450$", \
117         "$5_\u2264N_\u226410$", "$1\u2264N_\u22645$", "$N_\u22640$"]
118 patches = [ Patch(color = colors[i], label = "{:s}".format(texts[i])) \
119             for i in range(len(texts)) ]
120 plt.legend(handles = patches, loc = 'lower_left', \
121         title = 'Students')
122
123 #END
124 plt.show()

```

Avis de l'enseignant : le résultat (figure 13.1) mérite largement les efforts consentis durant ce cours !

13.2 Hydrogéologie des lacs des Landes

Nous allons représenter les isopièzes de la nappe du Plio-Quaternaire des Landes au sud du bassin d'Arcachon, dans la région des étangs de Cazaux et de Parentis. Nous disposons du shapefile des isopièzes de la nappe Plio-Quaternaire de l'Aquitaine¹. Notre objectif sera de représenter ces isopièzes dans la région des lacs sur une image satellite de la région.

Nous utiliserons une Landsat ETM+ que nous irons chercher sur le WMTS de la NASA présenté précédemment. Dans ce dernier cas il est possible (à l'écran of course) de voir la précision de l'image qui augmente avec l'effet de grossissement.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 cm_16.py
5
6 Draw equipotential (isopieze map) lines of the Plio-quaternary aquifer
7 on top of a Landast ETM view of the Lakes of Landes south of Arcachon.
8 The ETM image comes from the earthdata web service
9
10 cartopy version
11 """
12
13 #libraries
14 import geopandas

```

1. Origine des données : SIGES aquitaine <http://sigesaqi.brgm.fr/-Les-isopiezes-.html>.

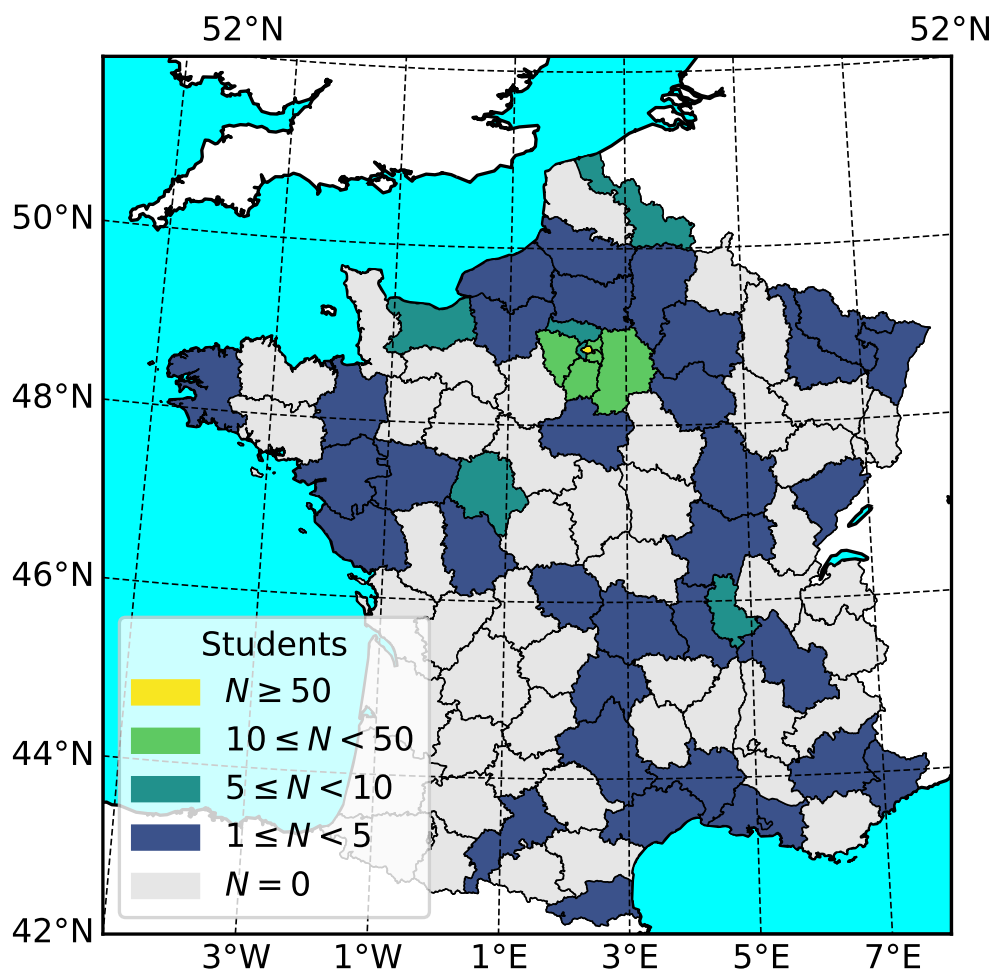


FIGURE 13.1 – Nombre agrégé d'étudiants par département un an avant leur arrivée à l'UFR STEP

```

15
16 import matplotlib as mpl
17 import matplotlib.pyplot as plt
18 from matplotlib import cm
19 from mpl_toolkits.axes_grid1.inset_locator import inset_axes
20
21 import cartopy.crs as ccrs
22 import cartopy.io.shapereader as shpreader
23
24 from owslib.wmts import WebMapTileService
25
26
27 def convert_shape():
28     #read the file
29     piezo = geopandas.read_file("../include/shapefiles/Aquitaine/Isopiezes_mio-plio-
        quaternaire.shp")
30
31
32     print(piezo.head())
33     print(piezo.crs)
34     print("└──────────────────┘")
35
36     #test to get rid of null or empty values in the record (else the conversion fails
        )
37     piezo = piezo[piezo['geometry'].notnull()]
38     #all is here the Plate carree definition
39     piezo2 = piezo.to_crs({'init': 'epsg:4326'})
40     print(piezo2.head())
41     print(piezo.crs)
42     print("└──────────────────┘")
43
44
45     #save the file
46     piezo2.to_file("./shapefiles/Aquitaine/Isopiezes_mio-plio-quaternaire_latlon.shp"
        ,encoding='utf-8')
47
48
49 def map_piezo():
50
51     # URL of NASA GIBS
52     URL = 'http://gibs.earthdata.nasa.gov/wmts/epsg4326/best/wmts.cgi'
53     wmts = WebMapTileService(URL)
54
55     # Layers for MODIS true color and snow RGB
56     layer = 'Landsat_WELD_CorrectedReflectance_Bands157_Global_Annual'
57
58     date_str = '1999-08-16'
59
60     # Plot setup
61     plot_CRS = ccrs.PlateCarree()
62     geodetic_CRS = ccrs.Geodetic()
63
64
65     x0, y0 = plot_CRS.transform_point(-1.43, 44.15, geodetic_CRS)
66     x1, y1 = plot_CRS.transform_point(-0.69, 44.68, geodetic_CRS)
67
68     fig = plt.figure(figsize = (20, 20))
69     ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
70
71
72     ax.set_xlim(-1.5,0)
73     ax.set_ylim(44,45)
74
75     ax.gridlines(draw_labels=True, xlocs=[-1.5,-1,-0.5,0], ylocs=[44,44.5,45],
        color='white', linestyle='--')
76     ax.coastlines(resolution='10m')

```

```

77
78     ax.add_wmts(wmts, layer, wmts_kwargs={'time': date_str})
79
80
81
82     # Open shapefile
83     reader = shpreader.Reader("../include/shapefiles/Aquitaine/Isopiezes_mio-plio
      -quaternaire_latlon.shp")
84     features = reader.records()
85
86
87     # retrieve the polylines and plot them using different colors
88     cmap = cm.get_cmap('YlOrRd')
89     for f in features:
90         c = cmap(float(f.attributes['COTE_mNGF'])/100) #100 is assumed to be
      the maximum piezometric head in the region
91         print(f.geometry)
92         #~ ax.plot(x, y, '-', linewidth = 2, color = c, label = info['
      COTE_mNGF'])
93         ax.add_geometries([f.geometry], edgecolor = c, facecolor='none',
      linewidth=2, label = f.attributes['COTE_mNGF'], crs=plot_CRS)
94
95
96     # Print some information
97     x, y = (-1.1681992, 44.45807)
98     ax.text(x, y, "Etang_de_Cazaux", color = 'w', transform=plot_CRS)
99     x, y = (-1.1749838, 44.345)
100    ax.text(x, y, 'Etang_de_Parentis', color = 'w', transform=plot_CRS)
101    x, y = (-1.4, 44.7)
102    ax.text(x, y, u"Océan_atlantique", color = 'w', rotation = 90, fontsize = 30,
      transform=plot_CRS)
103    x, y = (-1.184541, 44.668606)
104    ax.text(x, y, "Bassin_d'Arcachon", color = 'w', transform=plot_CRS)
105
106
107    axins = inset_axes(ax,
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128    map_piezo()

```

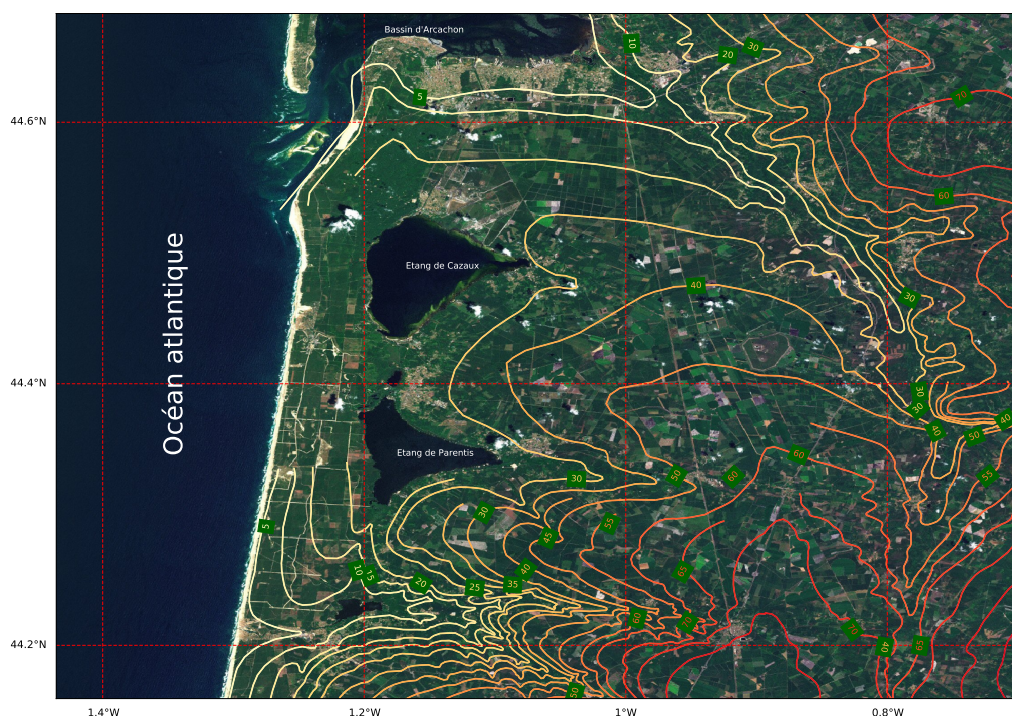


FIGURE 13.2 – Hydrogéologie des bassins des étangs de Cazaux et de Parentis. Image satellite : [ESRI World Imagery](#). Hydrogéologie : [SIGES Aquitaine](#).

13.3 Lacs des Landes, seconde manière

La seconde façon d'utiliser le même jeu de données consiste à interpoler² les isopièzes du shapefile sur une grille dans python puis d'utiliser cette grille pour représenter les izopièses en utilisant les fonctions classiques de matplotlib comme `contourf` par exemple. Cette seconde méthode présente deux avantages et un inconvénient ; Les avantages sont (1) une écriture plus simple, (2) la création d'une grille de données que l'on peut ensuite utiliser pour calculer des lignes de courant par exemple (figure 13.3). Le principal inconvénient c'est l'interpolation en elle même, notamment aux limites de la grille. Il faut donc faire des tests et attention au résultat. Afin d'éviter ces effets on utilise la méthode `griddata` de la bibliothèque `matplotlib.mlab`. Cette méthode, moins puissante que la méthode homonyme de `scipy`, présente l'avantage de ne pas extrapoler au-delà du polygone englobant les points de mesure. Elle retourne donc une grille masquée (masked array). Enfin notons que le gradient est calculé sur les coordonnées lat, lon et non sur les coordonnées projetées dans un système métriques. Tant que nous ne souhaitons que représenter les lignes de courants ce n'est pas grave. Pour un calcul de vitesses ou de débit par contre il faudra passer par des coordonnées métriques.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 cm_17.py
5
6 Purpose :
7 read shapefile
8 regrid equipotential head lines
9 calculate streamlines
10 plot
11
12 cartopy version
13 """
14 import geopandas
15
16 import matplotlib as mpl
17 import matplotlib.pyplot as plt
18 from matplotlib import cm

```

2. Il s'agit dans les faits d'une ré-interpolation car les courbes isopèzes sont déjà des contours tracés à partir de mesures ponctuelles de piézométrie.

```

19 from matplotlib.mlab import griddata # this one not the scipy one
20
21 from mpl_toolkits.axes_grid1.inset_locator import inset_axes
22
23 import cartopy.crs as ccrs
24 import cartopy.io.shapereader as shpreader
25
26 from owslib.wmts import WebMapTileService
27
28 import numpy as np
29
30 # open shapefile
31 deps = geopandas.read_file("../include/shapefiles/Aquitaine/Isopiezes_mio-plio-
      quaternaire_latlon.shp")
32
33 print( deps.head() )
34
35 X = []
36 Y = []
37 Z = []
38 for i in range(len(deps["geometry"])):
39
40     try:
41         #print deps["geometry"][i].geom_type
42         if deps["geometry"][i].geom_type == 'LineString':
43             x, y = deps["geometry"][i].coords.xy
44             z = deps['COTE_mNGF'][i]*np.ones(len(x))
45             X = np.hstack((X, x))
46             Y = np.hstack((Y, y))
47             Z = np.hstack((Z, z))
48
49         elif deps["geometry"][i].geom_type == 'MultiLineString':
50             for line in deps["geometry"][i]:
51                 x, y = line.coords.xy
52                 z = deps['COTE_mNGF'][i]*np.ones(len(x))
53                 X = np.hstack((X, x))
54                 Y = np.hstack((Y, y))
55                 Z = np.hstack((Z, z))
56
57     except:
58         print( i )
59         pass
60
61
62 xs = np.linspace(min(X), max(X), 400)
63 ys = np.linspace(min(Y), max(Y), 400)
64 xx, yy = np.meshgrid(xs, ys)
65 zz = griddata(X, Y, Z, xs, ys, interp = 'linear') # grid data this method does NO
      EXTRAPOLATION which leads to a masked array
66
67 v, u = np.gradient(zz)# calculate gradient BEWARE lines (y-direction) first, columns
      (x-direction) second
68
69
70 # Define figure size and basemap
71 # URL of NASA GIBS
72 URL = 'http://gibs.earthdata.nasa.gov/wmts/epsg4326/best/wmts.cgi'
73 wmts = WebMapTileService(URL)
74
75 # Layers for MODIS true color and snow RGB
76 layer = 'Landsat_WELD_CorrectedReflectance_Bands157_Global_Annual'
77
78 date_str = '1999-08-16'
79
80 # Plot setup
81 plot_CRS = ccrs.PlateCarree()

```

```

82 geodetic_CRS = ccrs.Geodetic()
83
84
85 x0, y0 = plot_CRS.transform_point(-1.43, 44.15, geodetic_CRS)
86 x1, y1 = plot_CRS.transform_point(-0.69, 44.68, geodetic_CRS)
87
88 fig = plt.figure(figsize = (20, 20))
89 ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
90
91
92 ax.set_xlim(-1.5,-0.5)
93 ax.set_ylim(44,45)
94
95 ax.gridlines(draw_labels=True, xlocs=[-1.5,-1,-0.5,0], ylocs=[44,44.5,45], color='
    white', linestyle='--')
96 ax.coastlines(resolution='10m')
97
98 ax.add_wmts(wmts, layer, wmts_kwargs={'time': date_str})
99
100 c = ax.contourf(xx, yy, zz, [10, 20, 30, 40, 50, 60, 70, 80], alpha = 0.4, transform=
    plot_CRS)
101 ax.contour(xx, yy, zz, [10, 20, 30, 40, 50, 60, 70, 80], transform=plot_CRS)
102 ax.streamplot(xx, yy, -u, -v, density = 1, color = 'red', arrowsize = 2.5, transform=
    plot_CRS) # streamlines  $\mathbf{U}_{\text{Darcy}}$   $\propto - \mathbf{grad}(h)$ 
103 cb = plt.colorbar(c, shrink=0.8)
104 cb.set_label("Hauteur piézométrique")
105
106 # Print some information
107 x, y = (-1.1681992, 44.45807)
108 ax.text(x, y, "Etang de Cazaux", color = 'w')
109 x, y = (-1.1749838, 44.345)
110 ax.text(x, y, 'Etang de Parentis', color = 'w')
111 x, y = (-1.4, 44.7)
112 ax.text(x, y, u"Océan atlantique", color = 'w', rotation = 90, fontsize = 30)
113 x, y = (-1.184541, 44.668606)
114 ax.text(x, y, "Bassin d'Arcachon", color = 'w')
115
116
117 # Print some information
118 x, y = (-1.1681992, 44.45807)
119 ax.text(x, y, "Etang de Cazaux", color = 'w', transform=plot_CRS)
120 x, y = (-1.1749838, 44.345)
121 ax.text(x, y, 'Etang de Parentis', color = 'w', transform=plot_CRS)
122 x, y = (-1.4, 44.7)
123 ax.text(x, y, u"Océan atlantique", color = 'w', rotation = 90, fontsize = 30,
    transform=plot_CRS)
124 x, y = (-1.184541, 44.668606)
125 ax.text(x, y, "Bassin d'Arcachon", color = 'w', transform=plot_CRS)
126
127 # plt.savefig("../figures/cm_shp2grid.pdf",bbox_inches='tight')
128 plt.show()

```

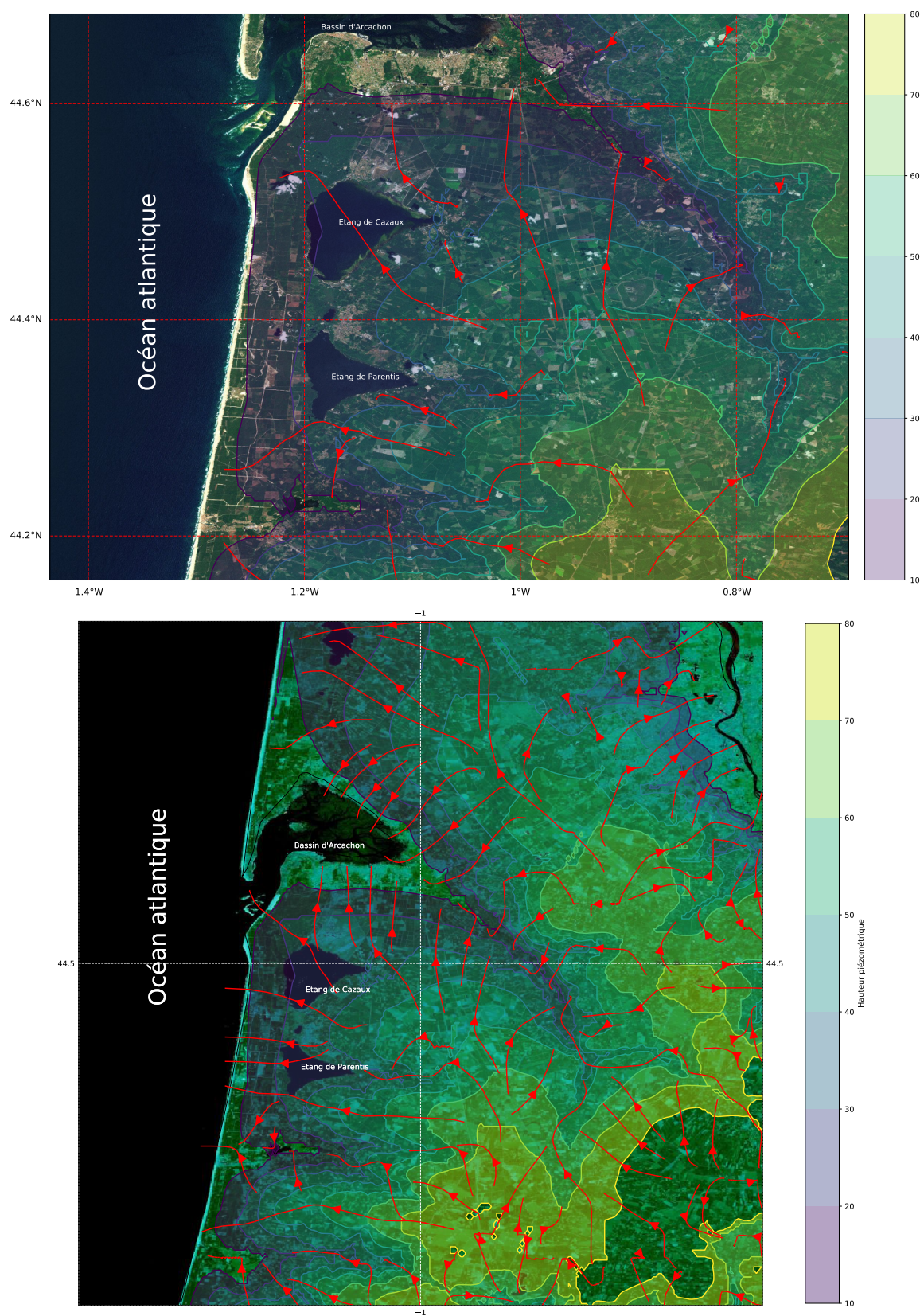



FIGURE 13.3 – Hydrogéologie des bassins des étangs de Cazaux et de Parentis 2ème version. En haut basemap, en bas cartopy

Chapitre 14

Faire des cartes en ligne

14.1 Introduction : Leaflet et Folium

La librairie `folium` est ce qu'on appelle un «wrapper» (un interface) qui permet, tout en programmant en python, d'accéder aux fonctionnalités d'une librairie javascript nommée `leaflet`. Ce que nous avons vu précédemment notamment sur l'usage des tuiles va nous être très utile dans ce qui suit. L'intérêt des cartes interactives et mises en ligne croit avec l'usage systématique d'internet. Il n'est donc pas inutile de savoir coder une carte web interactive et y mettre des données. C'est l'objet de ce chapitre.

14.2 Le fond de carte

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Mon May 18 09:16:23 2020
5
6  @author: metivier
7
8  folium maps
9
10 simple map with IPGP places in the world
11
12 """
13 import folium
14
15 #create the map
16 macarte = folium.Map(location=[25,2.4], zoom_start=3)
17
18 #add markers
19 folium.Marker([48.845366, 2.358222], popup="<img src='../include/images/cuvier.jpg'
    width='200px' /><b>Institut de Physique du globe de Paris</b><br>Main Cuvier
    Campus").add_to(macarte)
20 folium.Marker([48.827624, 2.380330], popup="<img src='../include/images/lamarckA.jpg'
    width='200px' /><img src='../include/images/lamarckB.jpg' width='200px' /><b>
    Institut de Physique du globe de Paris</b><br>UP Campus").add_to(macarte)
21 folium.Marker([15.979679, -61.702719], popup="<img src='../include/images/ovsg.jpg'
    width='200px' /><b>Institut de Physique du globe de Paris</b><br>Guadeloupe
    Observatory").add_to(macarte)
22 folium.Marker([14.734571, -61.159924], popup="<img src='../include/images/ovsm.jpg'
    width='200px' /><b>Institut de Physique du globe de Paris</b><br>Martinique
    Observatory").add_to(macarte)
23 folium.Marker([-21.208519, 55.571905], popup="<img src='../include/images/piton.jpg'
    width='200px' /><b>Institut de Physique du globe de Paris</b><br>La Réunion
    Observatory").add_to(macarte)
24 folium.Marker([48.025666, 2.260932], popup="<img src='../include/images/chambon.jpg'
    width='200px' /><b>Institut de Physique du globe de Paris</b><br>Chambon le
    forêt Observatory").add_to(macarte)
25
```

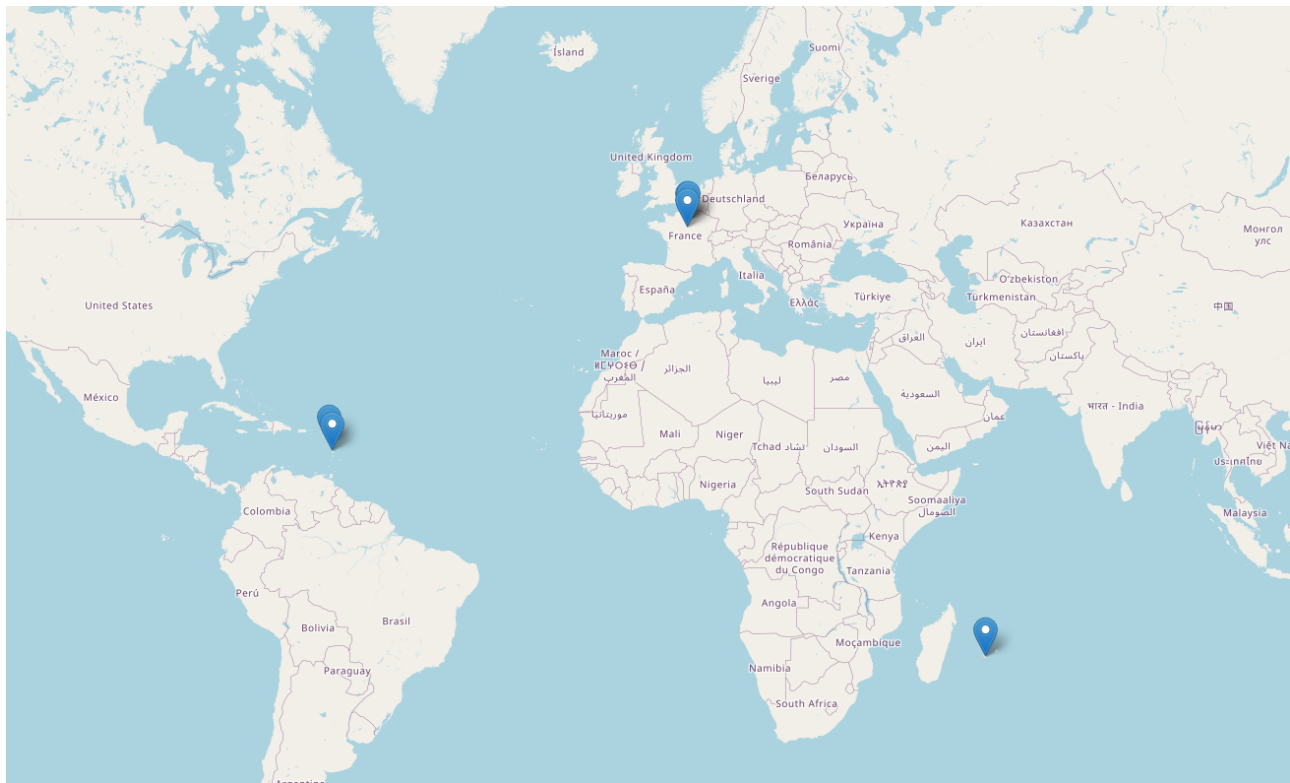


FIGURE 14.1 – Position des différents sites de l’IPGP.

```
26 #save to html file
27 macarte.save('f_1.html')
```

14.3 La gestion des tuiles

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Mon May 18 09:16:23 2020
5
6 @author: metivier
7
8 folium maps
9
10 Exemple of map made of wms server layers
11 here openstreet map is not the base layer
12
13 """
14 import folium
15
16 #create the map
17 macarte = folium.Map(location=[25,2.4], zoom_start=3, tiles=None)
18
19 URL = 'https://gibs.earthdata.nasa.gov/wms/epsg4326/best/wms.cgi'
20 Layer = 'ASTER_GDEM_Greyscale_Shaded_Relief'
21
22 myLayer = folium.raster_layers.WmsTileLayer(URL, Layer, name='ASTER_GDEM', transparent=
23     False, overlay=False)
24 myLayer.add_to(macarte)
25
26 Layer = 'GPW_Population_Density_2020'
27 myLayer = folium.raster_layers.WmsTileLayer(URL, Layer, name='Pop_Density', transparent=
28     True, overlay=True, opacity=0.5)
29 myLayer.add_to(macarte)
```



FIGURE 14.2 – Zoom sur une carte composée de trois tuiles superposées : un DEM ASTER en grisé, une carte de densité de population (2020) et le fond OpenStreetMap.

```

28
29 mytile=folium.raster_layers.TileLayer(overlay=True, opacity=0.6)
30 mytile.add_to(macarte)
31
32
33 folium.LayerControl().add_to(macarte)
34
35 #save to html file
36 macarte.save('f_2.html')
```

14.4 Ajouter des données : les séismes des 7 derniers jours

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Mon May 18 09:16:23 2020
5
6 @author: metivier
7
8 folium maps
9
10 Earthquakes of the last seven days plotted on top of three layers
11 ASTER GDEM Shaded Relief
12 Population density
13 Open street map tiles
14
15 """
16 import folium
17 import requests
18
19 import matplotlib.cm as cm
20 from matplotlib.colors import rgb2hex
21 from datetime import date, timedelta, datetime
22
23
24
25 #create the map
26 macarte = folium.Map(location=[25,2.4], zoom_start=3, tiles=None)
```

```

27
28 URL = 'https://gibs.earthdata.nasa.gov/wms/epsg4326/best/wms.cgi'
29 Layer = 'ASTER_GDEM_Greyscale_Shaded_Relief'
30
31 myLayer = folium.raster_layers.WmsTileLayer(URL, Layer, name='ASTER_DEM', transparent=
    False, overlay=False)
32 myLayer.add_to(macarte)
33
34 Layer = 'GPW_Population_Density_2020'
35 myLayer = folium.raster_layers.WmsTileLayer(URL, Layer, name='Pop_Density', transparent=
    True, overlay=True, opacity=0.5)
36 myLayer.add_to(macarte)
37
38 mytile=folium.raster_layers.TileLayer(overlay=True, opacity=0.6)
39 mytile.add_to(macarte)
40
41
42 folium.LayerControl().add_to(macarte)
43
44 #get the data
45 time_interval = 7 #decide the time interval (in days) you wish to search in the
    catalogue
46 today = date.today() #get today's date
47 startday = today - timedelta(days=time_interval) # get the starting date of seach
48
49
50
51 et=today.strftime("%Y-%m-%d")
52 st=startday.strftime("%Y-%m-%d")
53 minmag='1'
54
55 turl = "https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&starttime=%s&
    endtime=%s&minmag=%s" % (st, et, minmag)
56
57 response = requests.get(turl)
58
59 data = response.json()
60
61 cmap = cm.get_cmap('coolwarm')
62
63
64 #plot earthquakes as circles with individual popup info boxes.
65 for f in data['features']:
66     mag = f['properties']['mag']
67     coord = f['geometry']['coordinates']
68     c=cmap(1-coord[2]/500.)
69     comment = "<p style= width:200px; >"
70
71     edic = f['properties']
72     comment += "<b>%s</b>: %s<br>" % ('Time', datetime.fromtimestamp(edic['time']
        ]/1000).strftime('%c'))
73     comment += "<b>%s</b>: %s<br>" % ('Place', edic['place'])
74     comment += "<b>%s</b>: %s<br>" % ('Magnitude', edic['mag'])
75     comment += "<b>%s</b>: %s<br>" % ('Depth (km)', coord[2])
76     comment += "<a href=%s>More Info</a><br>" % edic['url']
77     comment += "</p>"
78
79     folium.CircleMarker(
80         location=[coord[1], coord[0]],
81         radius=5+2**(mag-1), #radius = f(mag)
82         popup=comment,
83         color='black',
84         weight=1, # circle line width !
85         fill=True,
86         fill_color=rgb2hex(c)# color=f(depth)
87     ).add_to(macarte)

```

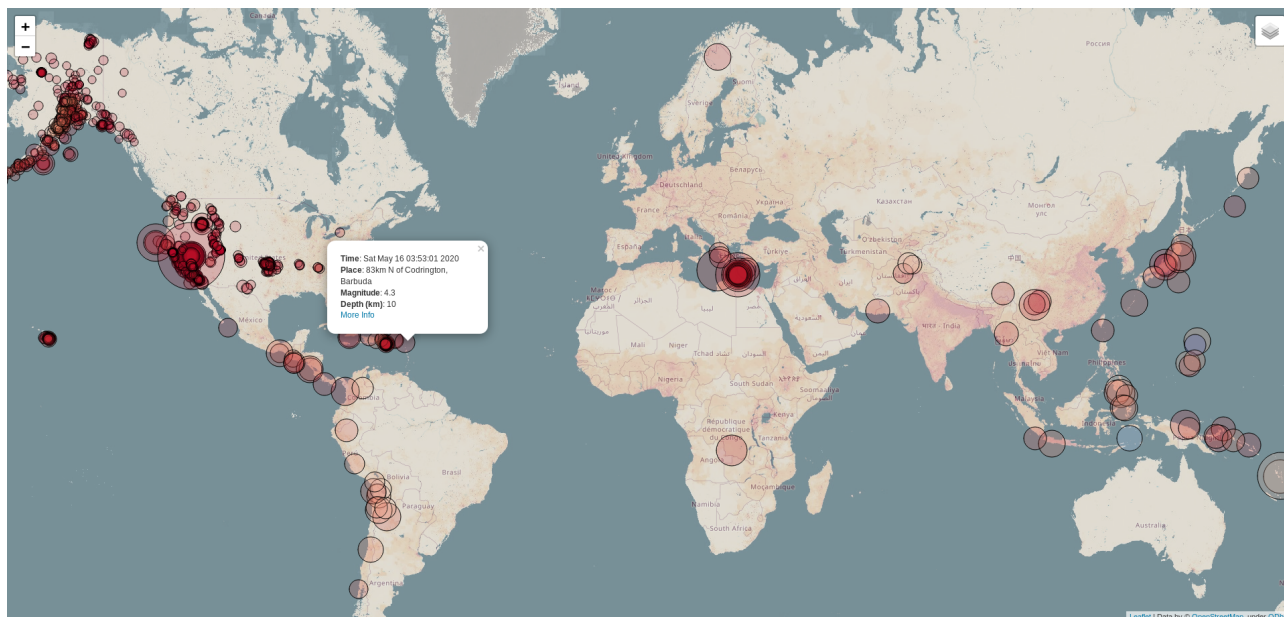


FIGURE 14.3 – Carte des séismes ($M \geq 1$) des 7 derniers jours. Source base de données de l'USGS

```
88
89
90 #save to html file
91 macarte.save('f_3.html')
```

14.5 Les clusters

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Mon May 18 09:16:23 2020
5
6 @author: metivier
7
8 folium maps
9
10 using Cluster Markers to plot the place where students where
11 the year before registering at IPGP
12
13 """
14
15 import folium
16 from folium.plugins import MarkerCluster
17
18 from sqlalchemy import create_engine
19 import pandas as pd
20 import numpy as np
21
22 #create the map
23 macarte = folium.Map(location=[46.5,2.5], zoom_start=6)
24
25 engine = create_engine("mysql://login:password@localhost/Parcours")
26
27 sql = "select cp, prenom from IdentiteTbl i inner join ParcoursTbl p on i.idetudiant =
28         p.idetudiant where annee=anneeuf-1 and pays='France'"
29
30 stud = pd.read_sql(sql, engine)
31 f=open('../include/data/laposte_hexasmal_1.csv')
```

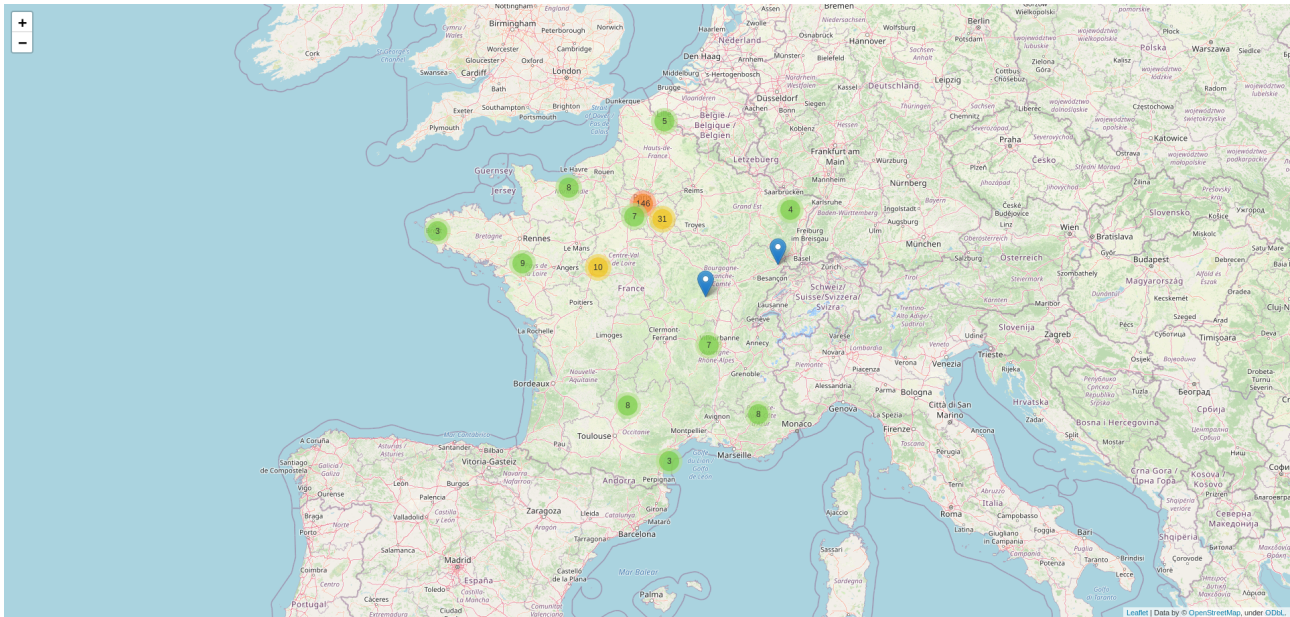


FIGURE 14.4 – Localité des étudiants un an avant leur arrivée en Licence ST

```

32 line=f.readline()
33 lines=f.readlines()
34
35 codep=[]
36 lat=[]
37 lon=[]
38 for l in lines:
39     d=l.split(',')
40     codep.append(d[2])
41     lat.append(d[5])
42     lon.append(d[6].strip('\n'))
43
44 data={'cp':codep, 'lat':lat, 'lon':lon}
45
46 cp = pd.DataFrame(data,columns=['cp','lat','lon'])
47 ccp= cp.drop_duplicates('cp')
48
49 merged=stud.set_index('cp').join(ccp.set_index('cp'))
50 merged=merged.dropna()
51
52 mc = MarkerCluster()
53 for index, row in merged.iterrows():
54     mc.add_child(folium.Marker(location=[row['lat'],row['lon']],popup=row['prenom']))
55
56 macarte.add_child(mc)
57
58 macarte.save('f_4.html')
59
60

```

14.6 GeoJson

Folium est une librairie qui permet de transformer vos instructions en instructions javascript de la librairie cartographique leafletjs. Le format standard d'échange et de gestion de données en javascript s'appelle JSON (Javascript Object Notation). Il possède deux versions adaptées aux SIG le GeoJSON et le TopoJSON qui prennent en charge les géométries. la différence entre ces deux librairies tient au fait que TopoJSON permet d'éviter la redondance des tracés. Ainsi, dans un fichier TopoJSON la frontière entre le Canada et les États-Unis ne sera enregistrée qu'une fois alors qu'elle le sera deux fois dans un fichier GeoJSON. Du fait de l'utilisation de JSON comme format de données il n'est pas possible de représenter directement un shapefile avec folium. Il

faut d'abord passer par une étape de conversion des données au format GeoJSON. Le script `ShapeTransform.py` montre une façon de procéder à partir des libraires `osgeo` et `json`. La fonction `shape_to_geojson` retourne un objet JSON et un DataFrame Pandas à partir d'un fichier shapefile

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Library to transform shapefiles
5  """
6  import osgeo.ogr
7  import json
8  import pandas
9
10
11 def shape_to_geojson(fname):
12     """
13     reads a shapefile and returns
14         1) a geosjson object
15         2) a pandas dataframe
16     """
17
18     #get the shapefile
19     shapefile = osgeo.ogr.Open(fname)
20     layer = shapefile.GetLayer(0)
21     layerdefn = layer.GetLayerDefn()
22     FieldList = []
23     FieldType = []
24     dfDic = {}
25     for i in range(layerdefn.GetFieldCount()):
26         FieldName=layerdefn.GetFieldDefn(i).GetName()
27         FieldList.append(FieldName)
28         dfDic[FieldName]=[]
29
30     numFeatures = layer.GetFeatureCount()
31
32     #initialize the Geojson dic
33     data={"type": "FeatureCollection", "features": []}
34
35     jd = json.JSONDecoder()
36
37     #loop through the data
38     for featureNum in range(numFeatures):
39         feature = layer.GetFeature(featureNum)
40
41
42         #first add to the Geojson dic
43         fdic={}
44         fdic["type"] = "Feature"
45         attributes = feature.items()
46
47         #create the properties dic
48         pdic={}
49         for key, val in attributes.items():
50             pdic[key] = val
51             fdic["properties"] = pdic
52
53         #create the geometry dic
54         geometry = feature.GetGeometryRef()
55         # we must first export to json then Transform into a dic
56         fdic["geometry"] = jd.decode(geometry.ExportToJson())
57
58         #add the feature to the featurecollection dic
59         data["features"].append(fdic)
60
61         #second add to the data columns
62         for Field in FieldList:
63             dfDic[Field].append(attributes[Field])

```



```

64         df = pandas.DataFrame(dfDic)
65
66     return json.dumps(data), df
67
68
69
70 if __name__ == '__main__':
71
72     data, df = shape_to_geojson('../map/include/shapefiles/
73     ne_50m_admin_0_countries/ne_50m_admin_0_countries.shp')
74     print(df.head())
75     print(df.columns)

```

Une fois cette conversion faite il devient alors possible de représenter nos données sur une carte en utilisant différentes méthodes de folium.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  @author: metivier
5
6  folium maps
7
8  Transform plot GeoJon polygons
9  and add information un Popups
10
11  """
12
13  import folium
14  import numpy as np
15  import pandas
16
17  from ShapeTransform import shape_to_geojson
18
19  def big_style(x):
20      """
21      samples tyle function
22      """
23      if x['properties']['POP_EST'] <= 1e6 :
24          stdic = {'fillColor': 'darkgreen', "color":'black', "opacity":0.4}
25      elif x['properties']['POP_EST'] <= 1e7 and x['properties']['POP_EST'] > 1e6:
26          stdic = {'fillColor': 'green', "color":'black', "opacity":0.4}
27      elif x['properties']['POP_EST'] <= 1e8 and x['properties']['POP_EST'] > 1e7:
28          stdic = {'fillColor': 'yellow', "color":'black', "opacity":0.4}
29      elif x['properties']['POP_EST'] <= 1e9 and x['properties']['POP_EST'] > 1e8:
30          stdic = {'fillColor': 'orange', "color":'black', "opacity":0.4}
31      elif x['properties']['POP_EST'] > 1e9:
32          stdic = {'fillColor': 'red', "color":'black', "opacity":0.4}
33      return stdic
34
35  macarte = folium.Map(location=[46.5,2.5], zoom_start=3)
36
37  data, df = shape_to_geojson( '../map/include/shapefiles/ne_50m_admin_0_countries/
38  ne_50m_admin_0_countries.shp' )
39
40  #add some information because it's a mess to get the legend
41  JsonLayer = folium.GeoJson(data,style_function=big_style,name='Population')
42  JsonPop = folium.features.GeoJsonPopup(fields=['NAME','POP_EST'],aliases = ["Country"
43  ,"Population"]).add_to(JsonLayer)
44  JsonLayer.add_to(macarte)
45
46  folium.LayerControl().add_to(macarte)
47  macarte.save('f_5.html')

```



FIGURE 14.5 – Population mondiale première version : GeoJson + GeoJsonPopup

14.7 Choroplèthes

Les Choroplèthes (Choropleth) sont des cartes dont certaines régions sont teintées en fonction des valeurs prises par une variable. Une carte de France dont les départements sont coloriés en fonction, par exemple, de leur population est une Choroplèthe. La figure 14.5 que nous venons de faire est une Choroplète tout comme les figures 11.3 et 13.1. folium propose une fonction `Chloropleth` qui permet de représenter des données en provenance d'un `DataFrame` Pandas sur une géométrie `GeoJSON`. Le script suivant montre comment s'en servir. L'avantage de cette fonction est qu'elle gère aussi la représentation de la légende et qu'elle est assez simple à mettre en oeuvre.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  @author: metivier
5
6  folium maps
7
8  Plot Choropleth from a shapefile
9
10
11  """
12
13  import folium
14  import matplotlib.cm as cm
15  from matplotlib.colors import rgb2hex
16  import numpy as np
17  import pandas
18
19  from ShapeTransform import shape_to_geojson
20
21
22  macarte = folium.Map(location=[46.5,2.5], zoom_start=3)
23
24  data, df = shape_to_geojson( '../map/include/shapefiles/ne_50m_admin_0_countries/
25                          ne_50m_admin_0_countries.shp' )
26
27  # Populations extends over orders if magnitude so we need to
28  # prepare the data a little
29  df = df.fillna(value={'POP_EST': 1}) # get rid of nans

```

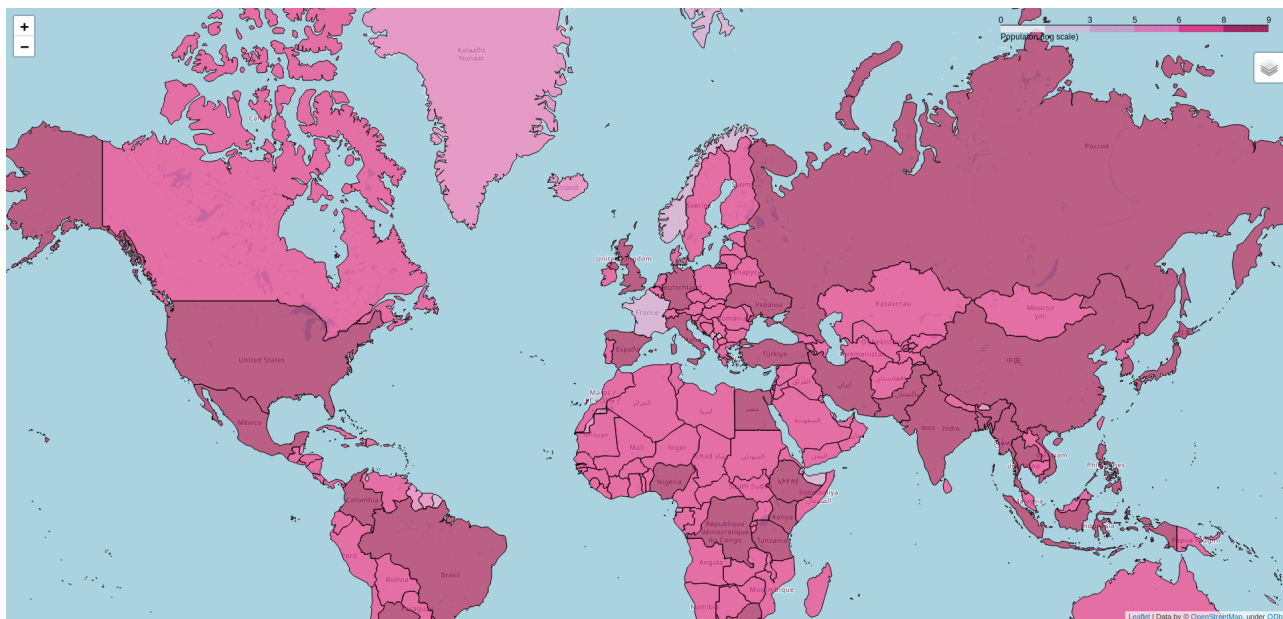


FIGURE 14.6 – Population mondiale seconde version ; Choropleths

```

30 df = df.replace({'POP_EST' : 0}, 1) # replace 0 values
31 df['POP_EST'] = np.log10(df['POP_EST']) # go to log10 scale
32
33 #In the end plot
34 cp = folium.Choropleth(data,df,columns=['ISO_A3','POP_EST'],key_on='feature.
    properties.ISO_A3',\
35     fill_color='PuRd',name='Population',legend_name='Populaton_(log_scale)').add_to(
        macarte)
36
37
38 folium.LayerControl().add_to(macarte)
39 macarte.save('f_6.html')

```

14.8 Ajouter des Traçés

Les deux exemples suivant montrent comment utiliser des données de position pour tracer des courbes. Dans le premier cas les données sont dans une base de données MySQL qui recense les ouragans de l'atlantique. Dans le second cas les données proviennent d'un shapefile dans lequel on trouve les isopièzes de la nappe des Landes.

14.8.1 Trajectoires d'ouragans

Les ouragans sélectionnés sont ceux de 2018. Attention si vous sélectionnez tout vous risquez d'avoir une carte très lourde...

```

1 #####
2 # librairies
3 #####
4
5 import folium
6 from sqlalchemy import create_engine
7
8 #####
9 #create the map
10 #####
11 macarte = folium.Map(location=[10,-20], zoom_start=3)
12
13 #####
14 # connexion
15 #####

```

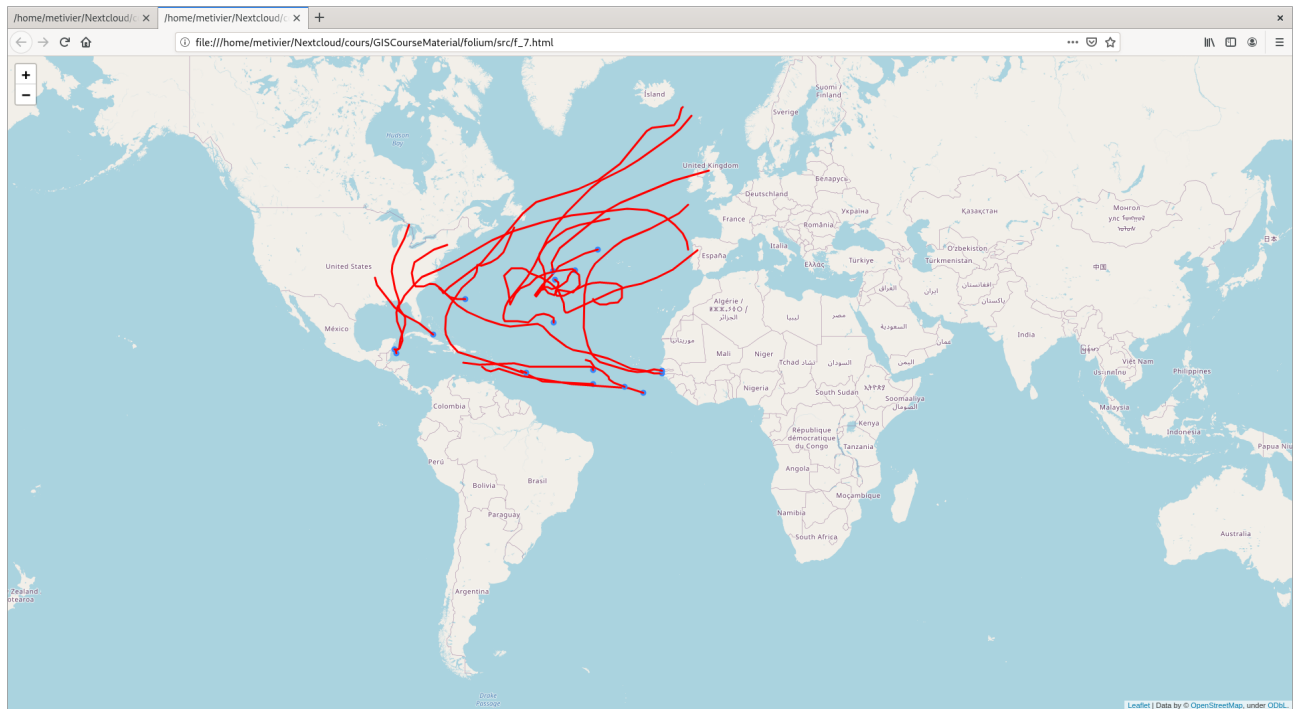


FIGURE 14.7 – Trajectoire des ouragans de 2018.

```

16 engine = create_engine("mysql://login:password@localhost/hurricanes")
17 conn = engine.connect()
18
19 sql = "select distinct code, name from reference where code regexp '2018'"
20 res = conn.execute(sql)
21
22 for line in res:.5
23     # print code
24     print( line[0] )
25     sql = "select latitude, longitude from data where code = '%s'"
26     pos = conn.execute(sql, (line[0]))
27     points = []
28     for p in pos:
29         if p[0][-1] == 'N':
30             lat = float( p[0][:-1] )
31         else:
32             lat = - float( p[0][:-1] )
33         if p[1][-1] == 'E':
34             lon = float( p[1][:-1] )
35         else:
36             lon = - float( p[1][:-1] )
37
38         points.append((lat,lon))
39
40         folium.CircleMarker( points[0] , radius = 3, popup = line[0] + '<br>' + line[1]).
41             add_to(macarte)
42         folium.PolyLine(points, color='red').add_to(macarte)
43
44 #save to html file
45 macarte.save('f_7.html')
```

14.8.2 Carte piézométrique

Deux petites fonctions sont créées afin de (1) faciliter la recherche des hauteurs piézométrique min et max et (2) inverser les coordonnées lon/lat des shapefiles en lat/lon pour folium.

```
1 #!/usr/bin/env python3
```

```
2 # -*- coding: utf-8 -*-
3 """
4 @author: metivier
5
6 folium maps
7
8 Plot Choropleth from a shapefile
9
10
11 """
12 #####
13 # Librairies
14 #####
15 import folium
16 import numpy as np
17 import cartopy.io.shapereader as shpreader
18 import branca.colormap as cm
19
20
21 #####
22 # extract extent of a shapefile attribute
23 #####
24
25 def get_extent(shapefile, attribute):
26     reader = shpreader.Reader(shapefile )
27     features = reader.records()
28
29     min_val = 9e19
30     max_val = -9e19
31     for feature in features:
32         min_val = min(min_val,feature.attributes[attribute])
33         max_val = max(max_val,feature.attributes[attribute])
34
35     return min_val, max_val
36
37 #####
38 # transform linestring to folium polyline ada
39 #####
40 def LineString_to_folium_dat( idat ):
41     odat = [(t[1], t[0]) for t in idat]
42     return odat
43
44 #####
45 # create the map
46 #####
47
48 macarte = folium.Map(location=[44.5,0], zoom_start=8)
49
50 #####
51 # read shapefiles
52 #####
53
54 filename = '../map/include/shapefiles/Aquitaine/Isopiezes_mio-plio-
55     quaternaire_latlon.shp'
56 reader = shpreader.Reader(filename)
57 features = reader.records()
58
59 #####
60 # create colormap
61 #####
62
63 cote_min, cote_max = get_extent(filename, 'COTE_mNGF')
64 print( cote_min, cote_max )
65 #dc = cote_max - cote_min
66 colormap = cm.linear.YlGnBu_05.scale(cote_min, cote_max)
```

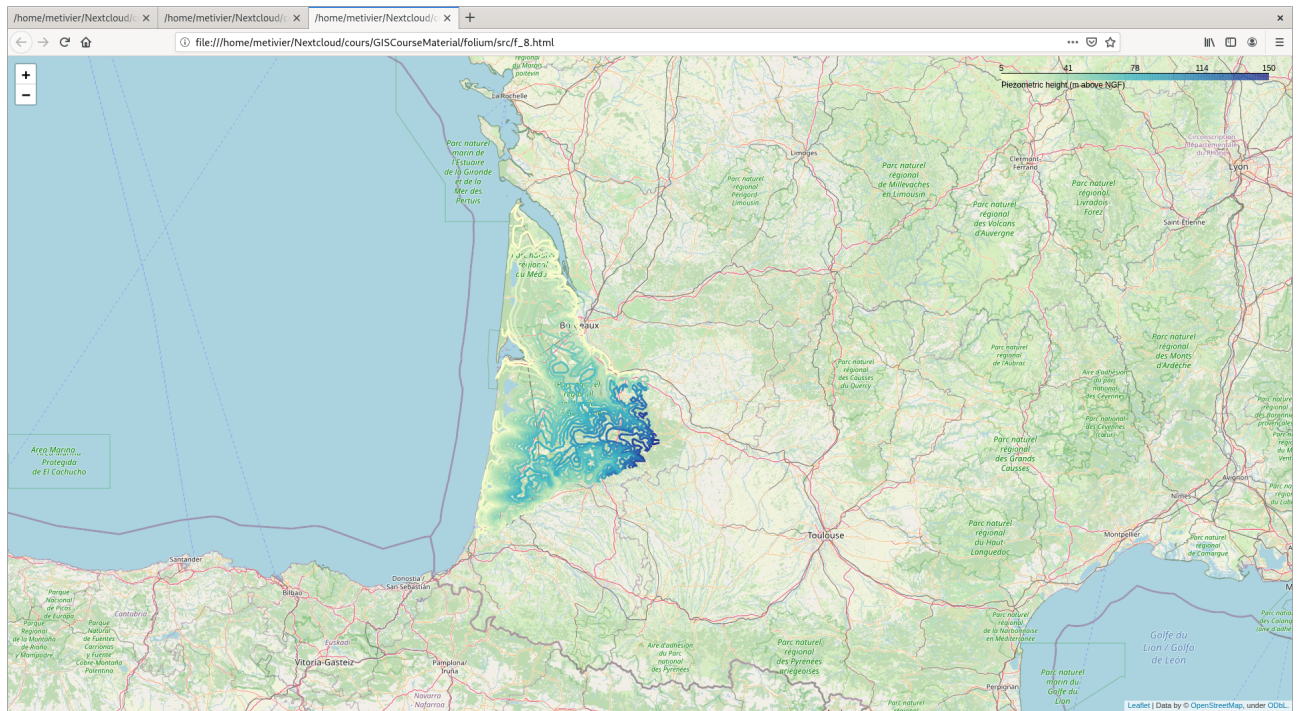


FIGURE 14.8 – Carte piézométrique de la nappe des Landes

```

67 colormap.caption = "Piezometric_height_(m_above_NGF)"
68 macarte.add_child(colormap)
69
70 #####
71 # loop to transform geometries into linestrings
72 # for folium.PolyLine then plot
73 #####
74 for feature in features:
75     cote = feature.attributes["COTE_mNGF"]
76     pp = "cote_NGF:_%f_m" % cote
77     col = colormap( cote )
78
79     if feature.geometry.geom_type == 'LineString':
80         dat = LineString_to_folium_dat( feature.geometry.coords )
81         folium.PolyLine(dat, popup = pp, color = col).add_to(macarte)
82     else:
83         for line in feature.geometry:
84             dat = LineString_to_folium_dat( line.coords )
85             folium.PolyLine(dat, popup = pp, color = col).add_to(macarte)
86
87
88 #####
89 # finished !
90 #####
91
92 macarte.save('f_8.html')

```


Annexes

Annexe A

Rappels Python

A.1 Introduction

Cette section regroupe quelques rappels de ce que vous devez savoir pour la suite de ce cours. Ce n'est pas un cours de python ! Pour cela consultez les cours d'E. Gayer en L2 ou le cours de python donné en sciences du vivant à l'université Paris Diderot ¹.

Python est un langage de programmation qui permet de faire réaliser par l'ordinateur des opérations très diverses des plus simples aux plus complexes. Comme tout langage il repose sur une syntaxe et un vocabulaire. Contrairement aux langues parlées, la syntaxe est strict (l'ordinateur n'essaiera pas de vous comprendre si vous faites des fautes de grammaire), et le vocabulaire de départ limité. mais ce vocabulaire peut être étendu à l'infini par l'intermédiaire de suites d'instructions regroupées dans ce qu'on nomme des fonctions ou des objets.

Python est un langage interprété, l'ordinateur exécute au fur et à mesure qu'il découvre ce qu'il lit. Les instructions que vous allez donner seront écrites de façon séquentielles. Ces instructions seront soit saisies directement dans l'interpréteur python, soit écrites dans un fichier texte (dont le nom finit avec l'extension .py) puis exécutées par l'interpréteur.

Comme pour MySQL nous allons utiliser un environnement simple et «brut» pour travailler. Celui-ci sera constitué de l'interpréteur et d'un éditeur de texte simple.

lancer l'interpréteur en tapant `python` dans une console. Vous verrez le prompt changer en `>>>`. Vous êtes «dans» python. Commençons.

Dans le programme ci-dessous nous demandons à python d'effectuer quatre opérations : deux créations de variables auxquelles nous affectons une valeur entière et deux calcul que nous enregistrons dans deux autres variables ².

```
1 >>>a=2 # affecte la valeur deux a une variable a
2 >>>b=3 # affecte la valeur trois a une variable b
3 >>>c=a+b # affecte la somme de a et b a une variable c
4 >>>d=a/b # affecte la division de a par b a une variable d
5 >>>c
6 5
7 >>>d
8 0
```

Ici les variables sont des nombres entier. Python effectuera donc des opérations entières, à moins que vous ne lui spécifiez. Par conséquent on aura $d = a/b = 0$. Pour passer d'un entier à un réel et obtenir la division de deux nombres réels il suffit de faire

```
1 >>>d=1.0*a/b
```

Multiplier par 1.0 transforme le calcul, initialement sur des nombres entiers, en calcul sur des nombres réels. On verra que l'on a d'autres façon de procéder par la suite.

Sur la syntaxe, 1 ligne = 1 instruction. On verra que le langage permet en fait de faire plus d'une chose par ligne de code mais pour l'instant restons en à cette idée simple et vrai dans la majorité des cas. Le vocabulaire quand à lui est composé de de «mots». Les mots réservés du langage sont en nombre limité :=, les opérateurs, `if`, `then`, `else`, `for`, `while` etc...

Considérons deux points sur un axe muni d'une origine et d'un repère. On veut calculer la distance qui sépare les deux points.

1. <https://python.sdv.univ-paris-diderot.fr>

2. Les fautes d'accentuations sont volontaires ! le paquet latex que j'utilise pour les listing ne les aime pas...

```

1 >>>x1=2
2 >>>x2=3
3 >>>if x1>x2:
4 ...     distance=x1-x2 # ne pas oublier d'indenter
5 ...else:
6 ...     distance=x2-x1
7 ...
8 >>> distance
9 1

```

On a introduit ce que l'on appelle une condition d'exécution. la distance est donnée par valeur absolue de la différence des deux positions. il faut donc tester quel est le nombre le plus grand et faire la «bonne» différence, telle que le résultat est positif. **On notera que les calculs exécutés sous conditions sont indentés par une tabulation.**

Deux défauts : x1 et x2 sont fixés et si on veut le faire pour deux autres valeurs il faut tout réécrire. Qu'à cela ne tienne les fonctions sont là pour nous

```

1 >>>def distance1D(x1,x2): # x1 et x2 sont des arguments
2 ...     if x1>x2:
3 ...         distance=x1-x2
4 ...     else:
5 ...         distance=x2-x1
6 ...     return distance
7 ...

```

On peut maintenant appeler notre fonction

```

1 >>>distance1D(3,8)
2 >>>5

```

Bon vous venez vaille que vaille d'écrire cinq lignes non ? Changeons de technique. lancer gedit et écrire le code dans un fichier que l'on exécutera ensuite. On en profitera pour utiliser une bibliothèque de fonctions fort utile qui simplifiera grandement l'écriture.

```

1 # une ligne de commentaire : mon beau programme
2 # fichier distance1D.py
3 import numpy as np
4 def distance1D(x1,x2):
5     # calcul de la distance sur un axe muni de son repere norme
6     return np.abs(x1-x2)

```

Bon ben j'ai le programme il est beau et je l'exécute comment ? une première façon consiste à rajouter l'appel de la fonction sous sa définition

```

1 # une ligne de commentaire : mon beau programme
2 # fichier distance1D.py
3 import numpy as np
4 def distance1D(x1,x2):
5     # calcul de la distance sur un axe muni de son repere norme
6     return np.abs(x1-x2)
7
8
9 distance1D(2,7)

```

puis dans une fenêtre vous exécutez

```
1 > python distance1D.py
```

Une seconde façon bien plus propre consiste à placer l'appel dans une condition «bizarre»

```

1 # une ligne de commentaire : mon beau programme
2 # fichier distance1D.py
3 import numpy as np
4 def distance1D(x1,x2):
5     # calcul de la distance sur un axe muni de son repere norme
6     return np.abs(x1-x2)
7
8
9 if __name__=='__main__':
10     distance1D(2,7)

```

Puis lancer le scripts de la même façon pour obtenir le même résultat. Idiot ? Voire.

A.2 Structure d'un programme

A.2.1 Libraries

Les bibliothèques sont des séries de fonctions ou de classes stockées dans des fichiers. ces fichiers peuvent à leur tour être stockés dans des répertoires. Vous devez déclarer les bibliothèques que vous devez utiliser en premier

```

1 #structure
2 #import needed libraries
3 # you have to choose one way OR the other not both
4
5 # first way NOT RECOMMENDED
6 from matplotlib.pyplot import *
7
8 # second way RECOMMENDED
9 import matplotlib.pyplot as py
10
11 # in the first case you call a function from the pyplot library by
12 x=arange(10)
13 #in the second case by
14 x=py.arange(10)

```

Les plus importantes librairies seront

- `numpy`, `scipy` : pour le calcul scientifique
- `matplotlib`: (`pylab`, `pyplot`) pour de très beaux graphiques
- `MySQLdb`, `pandas`, `sqlalchemy` : Pour se connecter à un serveur de données
- `basemap`, `pyshp`, `pyproj`, `shapely`, `geopandas` pour la cartographie

NOTE IMPORTANTE : `pylab` est une bibliothèque qui inclut toutes les fonctionnalités de traçage de `pyplot` + `numpy` il «ressemble» donc à `MATLAB`.

A.2.2 fonctions

```

1 #define the function
2 def square(x):
3     return x*x
4
5 #then use it Note the indentation !
6 y=square(2)
7 print y

```

A.2.3 classes

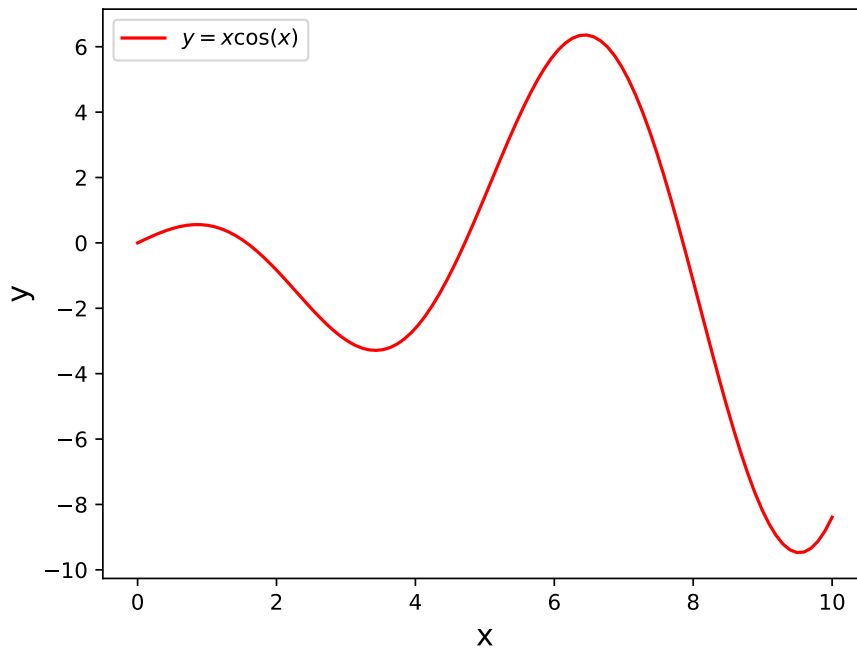
Les classes sont des objets abstraits en Python. Ce sont essentiellement des conteneurs qui ont des propriétés ou des fonctionnalités (ils peuvent effectuer des tâches).

```

1 class person:
2     def __init__(self, name="bobo", age="0"):
3         self.name=name
4         self.age=age
5         self.talk()
6
7     def speak(self):
8         print "my name is %s" % self.name
9         print "I am %s years old" % self.age
10
11
12 # use the class
13 levieux=person("Francois",47)
14 levieux.speak()

```

Quand vous écrivez `levieux=person("Francois",47)` vous créez une instance de la classe `person`, un peu à la façon dont vous créez un enregistrement dans une table. Cette instance (`Levieux`) hérite des propriétés et des fonctions (que l'on appelle des méthodes) de la classe `person`. De nombreuses bibliothèques de python sont organisées en classes.

FIGURE A.1 – $x \cos(x)$

`self` signifie que l'objet fait référence à l'instance de lui-même en cours. Les variables de l'objet sont ainsi notées `self.variable` et toute fonction de l'objet commence par faire référence à lui-même dans ses arguments.

A.3 Exemples

A.3.1 Premier graphique

Dans ce premier exemple, nous définissons un vecteur x qui va de 0 à 100 avec un pas de 0.1. Nous calculons ensuite $y = x \cos(x)$ puis nous représentons le résultat (Fig. A.1).

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_A1.py
4
5 plox xcos(x)
6 """
7 #import needed libraries
8 import matplotlib.pyplot as py
9
10 #define the data
11 x=py.linspace(0,10,100)
12 y=x*py.cos(x)
13
14 #plot the result
15 py.figure() #define the figure
16 py.plot(x,y,'r-') #make the plot
17 py.xlabel('x',fontsize=14) #add labels
18 py.ylabel('y',fontsize=14)
19 py.legend(["$y=x\cos(x)$"],loc="upper_left") # add legend including LaTeX commands
20 py.savefig("../figures/pm_A1.pdf",bbox_inches='tight') # save figure to file
21 py.show() # show the figure

```

A.3.2 Surface libre d'une nappe

Dans ce deuxième exemple, nous souhaitons tracer la surface libre, en régime stationnaire, d'une nappe libre dans un aquifère 1D de conductivité hydraulique K . L'eau entre à un flux q constant, au point $x = 0$ et de longueur L . La deuxième condition limite est une hauteur fixe de la nappe $h = 0$.

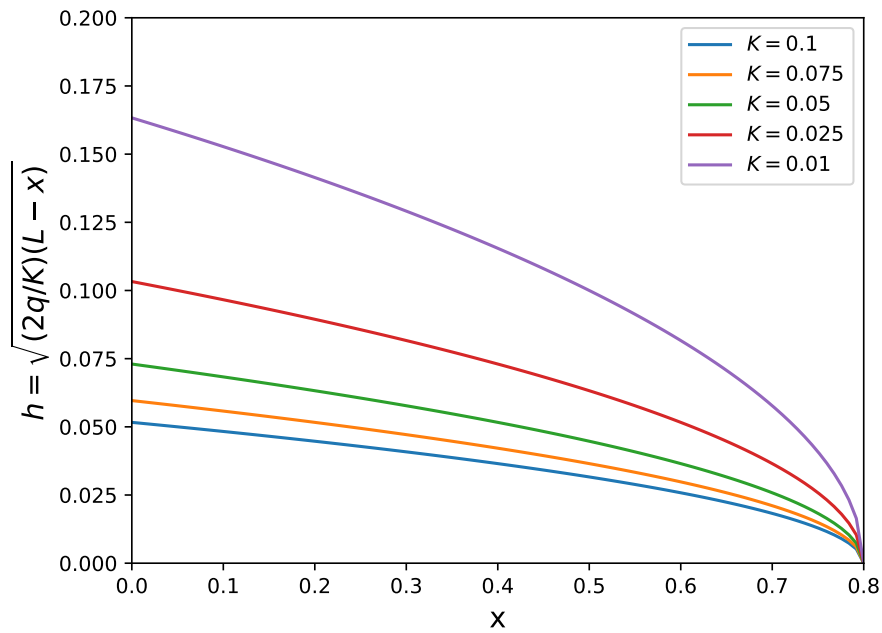


FIGURE A.2 – Profils d’une nappe en état stationnaire pour différentes conductivités hydrauliques.

Le problème est alors défini par

$$\nabla^2 h^2 = 0 \quad (\text{A.1})$$

$$-Kh\partial_x h(0) = q \quad (\text{A.2})$$

$$h(L) = 0 \quad (\text{A.3})$$

and the solution to (A.1), (A.2) and (A.3) is

$$h = \sqrt{\frac{2q}{K}(L-x)} \quad (\text{A.4})$$

Dans le script suivant, nous dessinons la forme de la nappe phréatique pour un aquifère expérimental de 0,8 m de longueur et pour différentes conductivités hydrauliques. Le résultat est tracé sur la figure A.2

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_A2.py
4
5 Solution of a steady 1D flow using Dupuit approximation over a length L
6 for different values of hydraulic conductivity K
7 Boundary conditions are a flux Q at 0 and zero depth y at 1.
8 """
9 # Import libraries
10 import matplotlib.pyplot as plt
11
12 # Define the problem
13 # constants
14 L = 0.8 # "true" length
15 Q = 1./6000 # discharge m2/s
16 K = [0.1, 0.075, 0.05, 0.025, 0.01] # hydraulic conductivity
17
18 # variable
19 x = plt.linspace(0, L, 100) # x value array
20
21 # Prepare the figure
22 plt.figure()
23 lgd=[] #legend
24
25 for k in K:

```

```

26     y = py.sqrt((2*Q/k)*(L-x)) # analytic solution for the water thickness
27     st = "$\u03a9K=%s\u03a9$" % (k) # add the conductivity to the legend
28     lgd.append(st)
29     py.plot(x, y)           #plot the result
30
31 # Figure labels, legend and axis
32 py.xlabel('x', fontsize = 14)
33 py.ylabel('$h=\sqrt{(2q/K)(L-x)}$', fontsize=14)
34 py.legend(lgd, loc = "upper\u0322right") # including LaTeX labels
35 py.axis([0, 0.8, 0, 0.2])
36
37 # save and display
38 py.savefig("../figures/pm_A2.pdf", bbox_inches = 'tight')
39 py.show()

```

A.4 L'utilisation des axes

le script suivant contient deux fonctions qui font appel à une librairie de matplotlib appelée `axes`. La première fonction fait littéralement la même chose que le script `pm_A1.py` vu au §A.3.1. La seconde plus élaborée permet de représenter deux fonctions $y = \cos(x)$ et $z = x^2$ sur la même figure à deux échelles différentes (figure A.3). À cette fin on représente les deux courbes sur deux systèmes d'axes différents. Ce genre de représentation peut s'avérer extrêmement utile. Elle nécessite de passer par la définition de deux axes pour une même figure.

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_A3.py
4
5 Using axes
6 """
7
8 #import needed libraries
9 import matplotlib.pyplot as py
10
11 def xcosx():
12     #define the data
13     x = py.linspace(0, 10, 100)
14     y = x*py.cos(x)
15     y2 = x*py.sin(x)
16     #plot the result
17     fig = py.figure() #define the figure
18     ax = fig.add_subplot(211) #create the axes and add a subplot
19     ax2 = fig.add_subplot(212)
20     ax.plot(x, y, 'r-') #make the plot
21     ax2.plot(x, y2, 'g-')
22     ax.set_xlabel('x', fontsize = 14) #add labels
23     ax.set_ylabel('y', fontsize = 14)
24     ax.legend(["$y=\u03a9x\u0322\cos(x)$"], loc = "upper\u0322left") # add legend including LaTeX
        commands
25     ax2.legend(["$y=\u03a9x\u0322\sin(x)$"], loc = "upper\u0322left") # add legend including LaTeX
        commands
26     py.show()
27
28
29
30 def two_axes():
31     #define the data
32     x = py.linspace(0, 100, 1000)
33     y = py.cos(x)
34     z = x**2
35     #plot the results
36     fig2 = py.figure() #define the figure
37     ax21 = fig2.add_subplot(111) #create the axes and add a subplot
38     ax22 = ax21.twinx() # create a new y-axis but keep the same x-axis
39     ax21.plot(x, y, 'r-', label = '$y=\u0322\cos(x)$') #make the plot
40     ax22.semilogy(x, z, 'g-', label = "$y=\u0322x^2$")

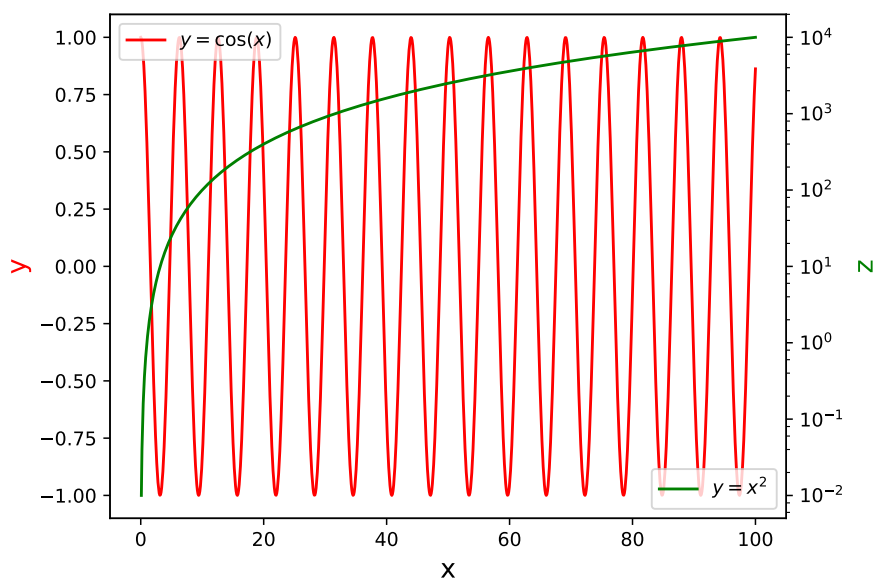
```

```

41 ax21.set_xlabel('x', fontsize = 14) #add labels
42 ax21.set_ylabel('y', color = 'r', fontsize = 14)
43 ax22.set_ylabel('z', color = 'g', fontsize = 14)
44 ax21.legend(loc = "upper_left") # add legend including LaTeX commands
45 ax22.legend(loc = "lower_right")
46 py.show() # show the figure
47
48 two_axes()

```

FIGURE A.3 – Doubler un axe avec axes.



Une figure peut-être considérée comme une page blanche. Les `axes` permettent de positionner un repère sur la page de la figure à l'endroit désiré comme le montre le script suivant

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 fig = plt.figure(figsize=(10, 5))
5 ax = fig.add_subplot(111)
6
7 x = np.linspace(0, 2*np.pi, 100)
8 y = np.cos(x)
9
10 ax.plot(x, y, 'r-')
11
12 ax2 = fig.add_axes([0.3, 0.7, 0.4, 0.2])
13 ax2.plot(x, np.sin(x), 'g-')
14 ax2.set_title("I can just do anything with axes!")
15
16 plt.savefig("../figures/pm_A3_1.pdf", bbox_inches = 'tight')
17 plt.show()

```

Ces deux programmes répondent à la question : Quelle est l'intérêt de passer par la définition d'un `axes`. Aucun si vous ne faites que des graphiques simples; obligatoire dès que vous souhaitez prendre le contrôle de votre figure et l'adapter réellement à vos besoins.

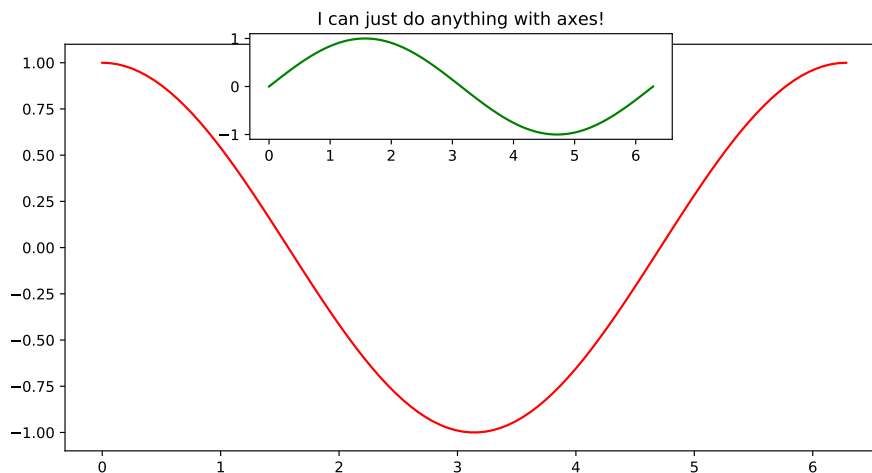
Enfin quand vous utilisez des `subplot()` comme

```

1 fig=figure()
2 ax1=fig.add_subplot(211)
3 ax2=fig.add_subplot(212)

```

vous vous situez entre deux eaux. `matplotlib` vous aide de cette façon à poser vos repères sur une grille ordonnée.

FIGURE A.4 – Positionnement des repères avec `axes`.

A.5 Collections et formes géométriques

Un exemple parmi d'autres mais important pour nous par la suite : l'utilisation de formes géométriques. Ces formes sont définies dans des classes, la classe `Circle` que nous allons utiliser par exemple, et font partie d'un ensemble appelé les `patches`. chaque forme peut être utilisée indépendamment ou regroupée dans des ensembles de formes que l'on appelle des collections de `patches` (`PatchCollection`). L'intérêt de regrouper les formes réside dans la possibilité offerte de leur attribuer des propriétés communes (même couleur etc...).

Nous allons dessiner des cercles (figure A.5)...

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_A4.py
4
5 Drawing Circle
6 """
7
8
9 import matplotlib.pyplot as plt
10 import matplotlib.patches as pch
11 import matplotlib.collections as col
12 from matplotlib.lines import Line2D
13
14 def first():
15     fig = plt.figure()
16     ax = fig.add_subplot(111)
17
18     # Create the circle
19     c = plt.Circle([25, 25], radius = 10, fc = 'r', ec = 'k', alpha = 0.5)
20     #add it to the figure
21     ax.add_artist(c)
22
23     # Set axis form
24     ax.axis('square')
25     # Fix the limits. Axes will not know the limits from the Circle instance
26     ax.set_ylim(0, 50)
27     ax.set_xlim(0, 50)
28
29     # Tricky Legend. Add a handle because legend can not get labels from a Circle (or
30     # a Rectangle or a Polygon)
31     hd = [Line2D([0], [0], marker = 'o', markersize = 14, markerfacecolor = 'r',
32     markeredgecolor = 'k', lw = 0, alpha = 0.5)]
33     ax.legend(hd, ['A red circle'])
34
35     # Save and show
36     plt.savefig("../figures/Circle.pdf", bbox_inches = 'tight')
37     plt.show()

```

```

36
37 def second():
38     fig = plt.figure()
39     ax = fig.add_subplot(111)
40
41     # Create the red circles
42     circles = []
43     for i in range(10):
44         c = pch.Circle([4*i,2*i], radius = 0.5*(i+1))
45         circles.append(c) # add the circle to a list of circles
46     # Create a collection of circles
47     ccol = col.PatchCollection(circles, edgecolor = 'k', facecolor = 'r', alpha =
48         0.5, label = 'red')
49     # Add them to the figure
50     ax.add_collection(ccol)
51
52     # s=Same for the green circles
53     circles = []
54     for i in range(10):
55         c = pch.Circle([2*i,4*i], radius = (i+1))
56         circles.append(c)
57     ccol = col.PatchCollection(circles, edgecolor = 'k', facecolor = 'g', alpha =
58         0.5, label = 'green')
59     ax.add_collection(ccol)
60
61     # Set the axis.
62     ax.axis('square')
63     ax.set_ylim(0,50)
64     ax.set_xlim(0,50)
65
66     # Tricky legend
67     # Create the handles using a Line2D object
68     hd = [Line2D([0], [0], marker = 'o', markersize = 14, markerfacecolor = 'r',
69         markeredgecolor = 'k', lw = 0, alpha = 0.5), Line2D([0],[0], marker = 'o',
70         markersize = 14, markerfacecolor = 'g', markeredgecolor = 'k', lw = 0,alpha =
71         0.5)]
72     # Create the legend calling the handles and adding labels.
73     ax.legend(hd, ['Red□circles','Green□circles'], labelspaceing = 1.2)
74
75     # Save and show
76     plt.savefig('../figures/Circles.pdf', bbox_inches = 'tight')
77     plt.show()
78
79 second()

```

La différence entre les deux fonctions vient de l'usage de ce que l'on nomme des **collections**. Celles-ci permettent de regrouper dans un même ensemble des formes aux propriétés équivalentes. Nous nous en servons lorsqu'il s'agira de colorier des polygones selon différents critères.

A.6 Légendes

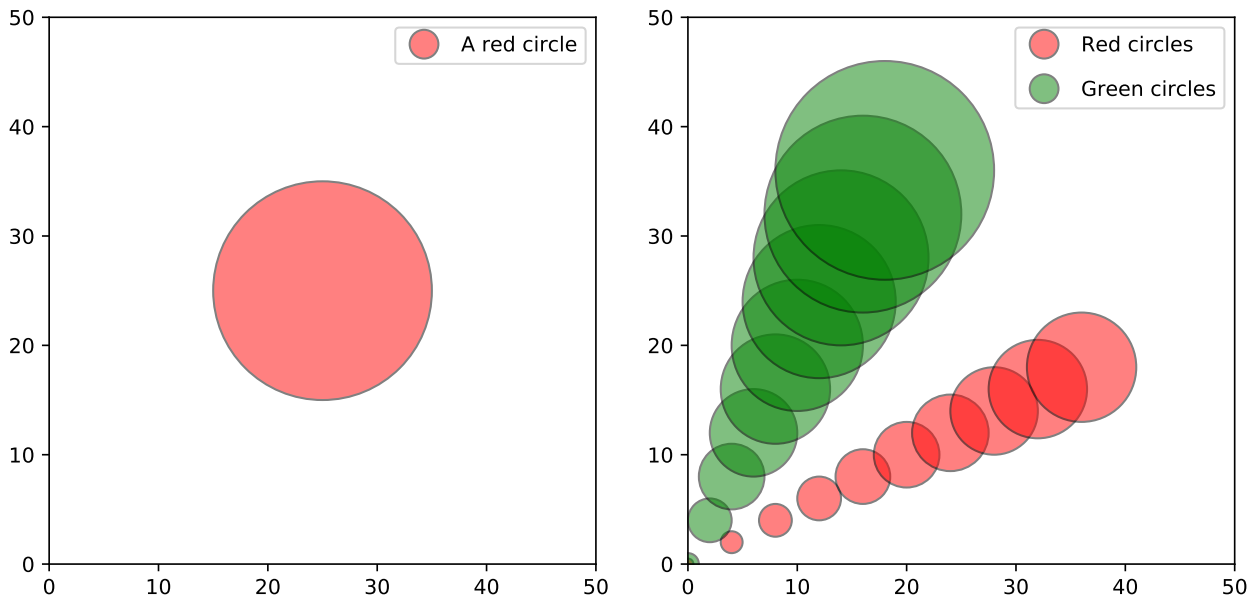
Comme pour tout en python, il y a plusieurs façons de représenter les légendes de figures, de la plus simple à la plus complexe. Voici quelques exemples. Comme toujours les arguments de la commande `legend` sont nombreux, et l'aide en ligne de matplotlib fort utile! Vous retrouverez dans la partie cartographie d'autres exemples montrant notamment l'utilisation de `colormap`.

```

1 # -*- coding: utf-8 -*-
2 """
3 pm_A5.py
4
5 Different ways to build a legend
6 """
7 #import needed libraries
8 import matplotlib.pyplot as plt
9 from matplotlib.lines import Line2D
10 import numpy as np

```

FIGURE A.5 – Utilisation des objets collections et patches



```

11
12 def leg_1():
13
14     """Adding labels"""
15
16     # Define the data
17     x = py.linspace(0, 10, 100)
18     y = x * py.cos(x)
19     z = x * py.sin(x)
20
21     py.figure() #define the figure
22     py.plot(x, y, 'r-', label = '$x\cos(x)$') #LABELS PLACED INSIDE THE PLOT
23     py.plot(x, z, 'g-', label = '$x\sin(x)$')
24
25     py.xlabel('x', fontsize = 14) #add labels
26     py.ylabel('y', fontsize = 14)
27
28     #LEGEND CALL
29     py.legend(loc = "upper_left") # add legend including LaTeX commands
30
31     py.show() # show the figure
32
33 def leg_2():
34
35     """Adding labels second version"""
36
37     # Define the data
38     x = py.linspace(0, 10, 100)
39     y = x*py.cos(x)
40     z = x*py.sin(x)
41
42     py.figure() # Define the figure
43     py.plot(x, y, 'r-')
44     py.plot(x, z, 'g-')
45
46     py.xlabel('x', fontsize = 14) # Add labels
47     py.ylabel('y', fontsize = 14)
48
49     # LEGEND CALL

```

```

50     py.legend(labels = ['$x\cos(x)$', '$x\sin(x)$'], loc = "lower_left") # add
      legend including LaTeX commands
51
52     py.show() # Show the figure
53
54 def leg_3():
55     """Retrieving axes, handles and labels"""
56
57     # Define the data
58     x = py.linspace(0, 10, 100)
59     y = x * py.cos(x)
60     z = x * py.sin(x)
61
62
63     fig = py.figure() #define the figure
64     ax = fig.add_subplot(111)
65     ax.plot(x, y, 'r-', label = '$x\cos(x)$')
66     ax.plot(x, z, 'g-', label = '$x\sin(x)$')
67
68     py.xlabel('x', fontsize = 14) #add labels
69     py.ylabel('y', fontsize = 14)
70
71     # LEGEND CALL
72     handles, labels = ax.get_legend_handles_labels()
73     py.legend(handles, labels, bbox_to_anchor = (0.5, 1.1), ncol = 2, loc = '
      center') # add legend including LaTeX commands
74
75     py.show() # show the figure
76
77 def leg_4():
78     """Phantom legend: very useful to represent collections and group labels"""
79
80
81     fig = py.figure() #define the figure
82     ax = fig.add_subplot(111)
83
84     # Create a phantom plot
85     for i in range(4):
86         py.scatter([], [], s = np.exp(i+1), edgecolor = 'k', facecolor = '
          none', label = str(i+1))
87
88     # Get the legend handles and labels
89     h, l = py.gca().get_legend_handles_labels()
90     # Add a new handle and label
91     h.append(Line2D([0], [0], color = 'r', lw = 3))
92     l.append('a red line!')
93
94     ax.legend(h, l, title = "Phantoms", labelspaceing = 2, borderpad = 2, frameon
      = True, ncol = 5, loc = 2)
95
96     py.show()

```

A.7 La fonction zip

cette fonction est importante car très utile lorsqu'on fait de la cartographie et qu'on veut associer ou dissocier des listes/tuples de données à des fins d'itération. Attention zip fonctionne différemment en python 2 et 3

```

1 # -*- coding: utf-8 -*-
2
3 print("original_tuples")
4 x1 = (0,1,2)
5 y1 = (1,2,3)
6 print(x1)
7 print(y1)
8

```

```
9 print("zip_function")
10 xy_iter = zip(x1,y1)
11 xy = list(xy_iter)
12 print(xy)
13
14 print("and_back")
15 x2,y2=zip(*xy)
16 print(x2)
17 print(y2)
18
19 print(*x2)
```

Annexe B

Bases de données supplémentaires

Ce chapitre permet de s'entraîner sur d'autres bases de données, environnementales ou non. Il vous permettra d'affermir votre maîtrise des requêtes.

B.1 La base GEMS

Nous allons travailler sur une base de données appelée **GEMS** (Global Environment Monitoring System) du nom du programme dont elle est issue. Contrairement au caractère général de son nom, cette base porte en fait sur les caractéristiques d'un échantillon des plus grands fleuves du monde. Elle comporte 6 tables dont les sujets sont

Nom de la table	Sujet
TblContinent	Continent auquel appartient un fleuve
TblExutoire	Exutoire vers lequel draine un fleuve
TblGemsInfoBassin	Informations géographiques sur les fleuves
TblFlux	Données physiques et flux
TblDissout	Chimie des éléments majeurs en solution

La base étant ici à vocation pédagogique les informations sont dans certains cas rudimentaires. Le découpage en tables différentes permet de différencier les sujets de chaque table. En effet je rappelle (encore !) que dans une base de données relationnelles une table se rapporte à un sujet bien défini. Le choix des sujets doit permettre de modifier les données d'une table (hormis certains champs particuliers) sans affecter les données des autres tables voire les autres champs d'une même table. On pourrait ainsi ajouter une description détaillée de chaque site d'étude dans la table `TblSiteEch`, ou des informations complémentaires sur les continents dans la table `TblContinent` sans avoir ni à modifier la structure de la base et encore moins à toucher les autres tables. Les clés primaires de la base **GEMS** sont :

Table	clé primaire
TblContinent	CodeContinent
TblExutoire	CodeExut
TblGemsInfoBassin	Bassin ¹
TblFlux	CodeFlux
TblDissout	CodeDissout

Une fois ces champs définis il est alors simple de relier les tables entre elles. En effet tout enregistrement de données de flux de la table `TblFlux` se rapportent à un bassin de la table `TblGemsInfoBassin`. Pour créer physiquement ce lien il suffit d'ajouter comme champs dans un enregistrement de la table `TblFlux` le champs `Bassin` de la table `TblGemsInfoBassin`. Ce champs n'a pas besoin d'être une clé (on peut avoir plusieurs enregistrements de flux pour une même rivière. Par contre il doit être identique (format et contenu) au champs de la table qu'il référence. Un tel champs est appelé une clé étrangère (Foreign Key) au sens où il fait référence à un table dans laquelle il a le statut de Clé primaire.

La figure [B.1](#) montre la façon dont les tables `TblGemsInfoBassin` et `TblExutoire` sont liées par un champs commun, qui a le statut de clé primaire dans le cas de la table exutoire et le statut de clé étrangère dans la table `TblGemsInfoBassin`. On notera que ce champs n'a pas le même nom. Peu importe en effet le nom seule la définition du champs et son contenu comptent. De plus un même code d'exutoire arrive fréquemment dans la table `TblGemsInfoBassin`. En effet plusieurs fleuves ont leur exutoire en Atlantique Nord par exemple.

1. En toute rigueur prendre le nom d'un bassin versant comme clé primaire n'est pas un bon choix. Pourquoi ?

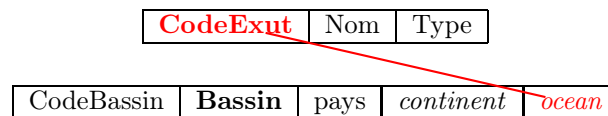


FIGURE B.1 – Lien entre deux tables

La structure de la base avec ses clés et liens est présentée dans la figure B.2. On notera que, dans le but de faciliter les requêtes que vous allez devoir exécuter, et vous éviter des requêtes demandant le recours à trois tables, un champs est redondant et ne devrait pas en toute logique figurer dans une des tables. Lequel et pourquoi ?

B.1.1 Requêtes simples sur une table

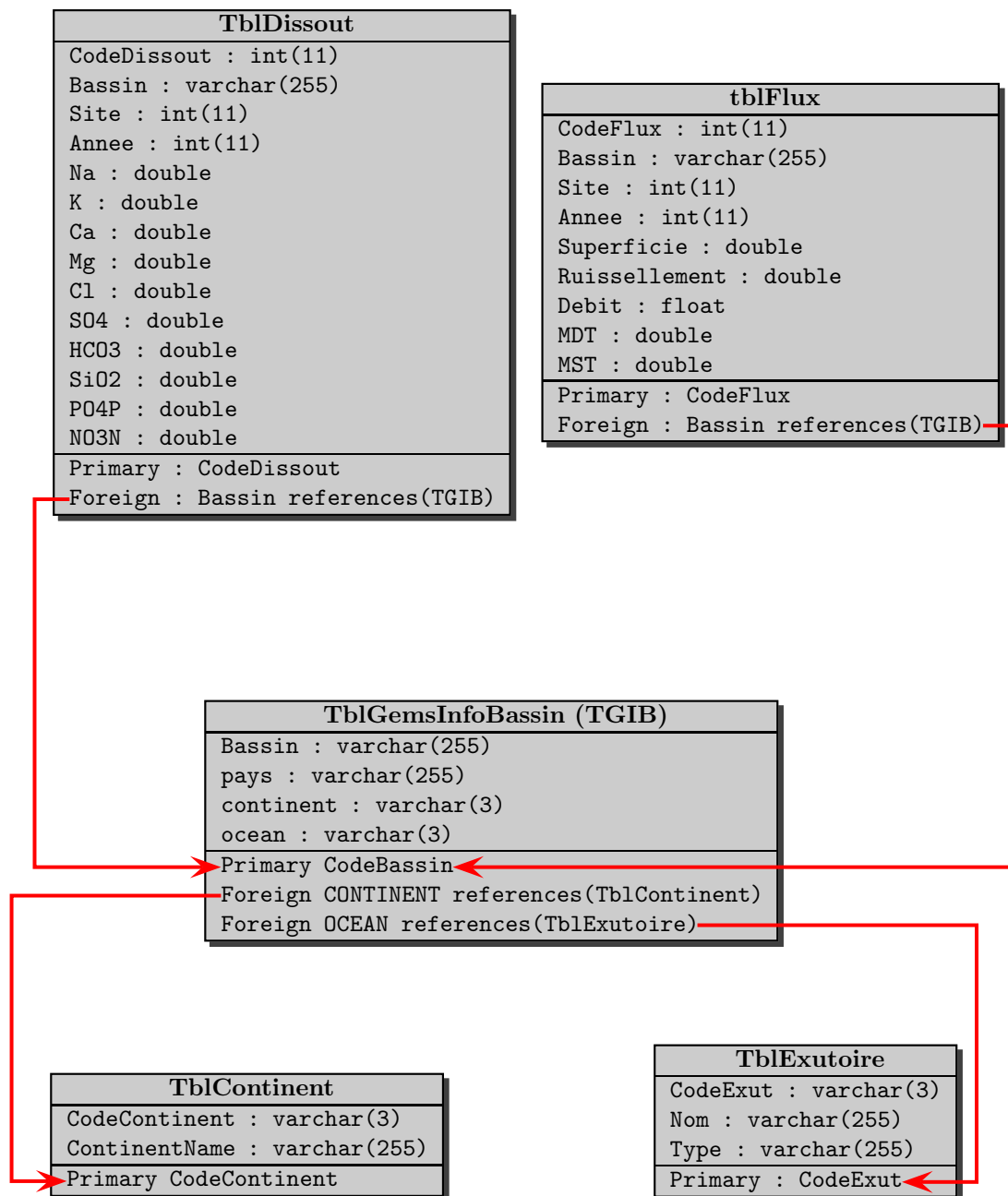
1. Combien d'enregistrements dans la table TLGemsInfoBassins ?
2. Combien d'enregistrements dans la table TblFlux ?
3. Combien d'enregistrements dans la table ContinentTbl ?
4. Quel est le débit en m³/s des fleuves de la table TblFlux classé par ordre décroissant de débit ?
5. Quel est le débit solide annuel des fleuves de la table TblFlux classé par ordre décroissant ?
6. Quel est le débit, le débit solide et le débit dissout (en unités SI) de la Chao Phraya ?
7. Quelle est la concentration en cations du Huang He ?
8. Quelle est la concentration en anions de la Seine ?
9. Quelle est la concentration en cations divalents de la Loire ?
10. Quel est le rapport entre concentration solide et concentration dissoute des fleuves de la base ?
11. Combien de ces fleuves ont un ratio supérieur à 1 ?
12. Combien de ces fleuves n'ont pas de ratio défini ?

B.1.2 Requêtes sur une ou plusieurs tables

1. Liste des bassins par continents ?
2. Liste des bassins d'Europe et d'Asie
3. Combien de bassins par continent ?
4. Liste des bassins et de leur Exutoire ordonnée par Exutoire ?
5. Combien de bassins par type d'exutoire ?
6. Combien de bassins en Amérique du sud ?
7. Quel pourcentage de la superficie du territoire couvrent les fleuves français de la base ?
8. Quel est le débit total annuel de chaque continent
9. Quelle est la charge sédimentaire annuelle (solide et dissoute) pour chaque continent en Mégatonnes par an ? Ordonnez le résultat par valeur décroissante de la charge
10. Quel est le flux de matière solide exporté en France en tonnes par an ?
11. Quel est le rapport flux solide, flux dissout pour chacune des rivières françaises
12. Quel est le rapport de ces flux pour chaque continent ? Ordonnez le résultat par rapport décroissant.
13. Combien de rivières dans chaque continent ont un débit liquide supérieur à 10000 m³/s ?
14. Quel est le débit du plus gros Fleuve d'Europe ?

B.1.3 Les requêtes emboîtées

1. Quel est le débit du plus gros fleuve d'Europe et quel est ce fleuve ? Deux solutions. Pour des raisons de précisions il faut éviter de faire les calculs dans la sélection du débit max. on peut le faire avec le débit annuel d'origine donné en valeurs entières sans problème.
2. Quelle est la contribution de l'Amazone au flux liquide, solide et dissout arrivant chaque année dans l'océan Atlantique ?



1

FIGURE B.2 – Schéma de la Base GEMS

B.2 Les Charts

La musique sauvera le monde. Un peu de détente donc... La base de données **Charts** ne contient qu'une table **SingleFrance** (table B.1) qui recense tous les singles du top 50 depuis sa création et jusque vers la fin 2018 (je la mets à jour quand j'ai le temps).

champs	type	description
year	int	année
week	int	semaine de l'année
rank	int	rang entre 1 et 50
artist	varchar(255)	nom de l'artiste ou du groupe
song	varchar(255)	nom de la chanson

TABLE B.1 – Structure de **SingleFrance**

B.2.1 Requêtes simples

1. Quelle est la liste des artistes et de leurs chansons présents dans la base classée par nom d'artiste décroissant et nom de chanson croissant.
2. Quelle est la liste des artistes qui ont été premiers au top 50
3. Comment se nomment les artistes dont le nom commence par «a»
4. Quels sont les artistes dont le nom contient «john»
5. Quelle est la liste des artistes ayant occupé la 1ère, la seconde ou la troisième position durant l'année 1991 ?
6. Quelle est la liste des chansons comprenant le mot "love" ou le mot "hate" dans leur titre ?
7. Quelle est la liste des chansons premières au top 50 entre 1985 et 1987 puis entre 1995 et 1997 ?
8. Quelle est la liste des artistes qui ont été premiers ou derniers en 1980, 1990, 2000 et 2010 ?
9. Combien d'enregistrements (de lignes) contient la table SingleFrance ?
10. Combien d'enregistrements (de lignes) contient la table SingleFrance pour chaque année du top50 ?
11. Combien d'artistes sont référencés dans la base ?
12. Combien de semaines chaque artiste a-t-il été présent dans le top 50 ?
13. Pour chaque chanson, combien de semaines un artiste a-t-il été dans le top 50 ?
14. Combien de chanson, dont le titre contient "love", ont-elle été premières au top 50
15. Combien d'artistes ont été derniers entre 2000 et 2010 ?
16. Pour un artiste de votre choix, et pour une de ses chansons, quelle est la trajectoire de celle-ci dans le top 50 ?
17. Toujours pour un artiste de votre choix, combien de semaines a-t-il (ou elle) occupé chacune de ses différentes positions au top 50 ?
18. Quels sont les artistes qui ont été plus de 100 semaines dans le top ?
19. Quels sont les artistes qui ont été plus de 10 semaines au premier rang ?
20. Quels sont les artistes qui ont été le plus au dernier rang ?

B.2.2 Requêtes emboîtées

1. Combien d'années chaque artiste a-t-il été présent au top 50 ?
2. Pour un artiste de votre choix, sur l'ensemble de sa production combien de semaines en moyenne occupe-t-il à une position quelconque ?
3. Même question pour tous les artistes ayant eu au moins 10 chansons au top.
4. Quelle est le temps moyen qu'un artiste passe à chacune des 50 positions du classement sachant son rang ?

Annexe C

Ressources internet

Liens valides le 16/09/2018.

C.1 L

L'aide en ligne est gigantesque. Je vous donne un site qui vous permet d'obtenir les informations fondamentales. Et sinon entrez votre question sur n'importe quel moteur de recherche et si vous ne trouvez pas la réponse c'est que soit vous ne savez pas vous y prendre, soit votre question commence à être réellement difficile, soit votre question est idiote (il faut toujours considérer ce point comme possible :=)).

1. Un site sur le SQL avec des cours, des liens etc... toujours en français! <https://sql.sh/>

C.2 Python

Idem l'aide en ligne est gigantesque. Entrez votre question dans un moteur de recherche et vous obtiendrez probablement une réponse. N'oubliez juste pas les quatre liens importants pour numpy, scipy, matplotlib et pandas

1. <http://www.numpy.org/>
2. <https://scipy.org/>
3. <https://matplotlib.org/>
4. <https://pandas.pydata.org/>

C.3 Basemap, cartopy et SIG

Là encore de nombreux sites mais qui reprennent presque tous les mêmes bases. En voici quelques uns que je trouve utiles.

1. Le tutoriel en ligne, très bien fait : <https://basemaptutorial.readthedocs.io/en/latest/>
2. Idem le site de cartopy est très bien fait et accessible <https://scitools.org.uk/cartopy/docs/latest/>
3. Quelques exemples de raster https://rabernat.github.io/research_computing/intro-to-basemap.html
4. Le cookbook ogr/gdal mais il n'est pas très friendly d'accès <https://pcjericks.github.io/py-gdalogr-cookbook/>
5. Un excellent cours sur les données vectorielles et raster et l'utilisation d'ogr et de gdal. Aride et pas de cartographie à proprement parler mais vraiment bien <https://www.gis.usu.edu/chrisg/python/2009/>
6. Des ressources intéressantes mais il faut chercher un peu. En français. <http://www.portailsig.org/>
7. Enfin une liste de bibliothèques spatiales en python https://github.com/SpatialPython/spatial_python