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-01877606v1

HAL Id: cel-01877606

https://hal.science/cel-01877606v1

Submitted on 20 Sep 2018 (v1), last revised 12 Sep 2021 (v3)

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

UFR STEP-IPGP.

20 septembre 2018

CC BY SA

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

3 Les requètes sur un table **19**
3.1 La base de données Parcelle . . . 19
3.2 Structure et chargement de la table météorologique : `imetos` . . . 19
3.3 Requêtes simples . . . 19
3.3.1 Sélectionnez un ou plusieurs champs d'une table . . . 19
3.3.2 Ordonner . . . 21
3.3.3 Distinguer . . . 21
3.3.4 Calculer . . . 21
3.3.5 Exporter . . . 21
3.4 Les conditions . . . 22
3.4.1 Comparaison . . . 22
3.4.2 Conditions multiples : utilisation des conjonctions . . . 22
3.5 L'emploi de fonctions . . . 23
3.5.1 Fonctions concernant le temps . . . 23
3.5.2 Fonctions de calculs . . . 23
3.5.3 Fonctions concernant les chaînes de caractères . . . 23
3.6 L'agrégation de données . . . 24
3.7 Applications . . . 24
3.8 Requêtes emboitées . . . 25
3.8.1 Principe . . . 25
3.8.2 Applications . . . 26
3.9 Vues . . . 26
3.10 Conclusion . . . 26

4 Tables multiples, clés et liens **27**
4.1 La base Parcours . . . 27
4.1.1 Structure de la base . . . 27
4.1.2 Implémentation . . . 28
4.1.3 Insérer les données . . . 29
4.2 Requêtes . . . 29

4.2.1 Patrons de recherche avec REGEXP . . . 29
4.2.2 Requêtes sur plusieurs tables . . . 30
4.2.3 Types de jointures . . . 31
4.2.4 Requêtes simples sur une ou deux tables . . . 32
4.2.5 Requêtes emboitées . . . 33

5 Utiliser Python et MySQL **35**
5.1 Introduction . . . 35
5.2 Examples . . . 35
5.2.1 Récupérer des données numériques . . . 35
5.2.2 Récupérer des données mixtes . . . 36
5.3 SQL DDL . . . 37
5.3.1 Traiter les erreurs . . . 40
5.3.2 OpenDB : a simple wrapper . . . 40
5.3.3 Précipitations : seconde implémentation . . . 41
5.4 Applications . . . 42
5.4.1 Cycles journaliers de température à Saint-Maur . . . 42
5.4.2 Direction des vents at Saint Maur . . . 43
5.5 Pandas et SQL . . . 45

II Cartographie **47**

6 La représentation cartographique avec `basemap` **49**
6.1 Introduction . . . 49
6.2 Première carte avec basemap . . . 49
6.3 Coordonnées, système de référence et projections . . . 50
6.3.1 Sélectionner une projection dans basemap . . . 50
6.3.2 Pyproj . . . 51
6.4 Les limites de la carte . . . 52
6.5 Une échelle svp . . . 53
6.6 Représenter des données . . . 54
6.6.1 Trajet entre deux deux capitales . . . 54
6.6.2 Carte de la sismicité (Scatter plot) . . . 55
6.6.3 Variation temporelle du champs magnétique (contour plot) . . . 57

7 Les shapefiles **59**
7.1 Formes et géométries . . . 59
7.2 OGR Data model . . . 59
7.3 Shapefile structure . . . 59
7.4 Représenter un shapefile . . . 61
7.4.1 Contours . . . 61
7.4.2 Colorier les polygones . . . 62
7.5 Les shapefiles comportant une projection . . . 64
7.6 Écrire des données dans un shapefile . . . 65

8 Les rasters **67**
8.1 Représenter un raster . . . 67
8.1.1 Landsat prétraitée de l'Aquitaine . . . 67
8.1.2 Topographie de la Guadeloupe . . . 70
8.2 Les images GeoTiff . . . 72
8.2.1 Histogramme . . . 73
8.2.2 Etirement des valeurs de pixels et fusion des canaux . . . 74
8.2.3 Joindre et découper . . . 75

9 Exemples d'applications **81**
9.1 Base des Parcours . . . 81
9.2 Hydrogéologie des lacs des Landes . . . 83
9.3 Lacs des Landes, seconde manière . . . 86

A Rappels Python **91**
A.1 Introduction . . . 91
A.2 Structure d'un programme . . . 93
A.2.1 Libraries . . . 93
A.2.2 functions . . . 93
A.2.3 classes . . . 94
A.3 Exemples . . . 94
A.3.1 Premier graphique : $y = x\cos(x)$ . . . 94
A.3.2 Surface libre d'une nappe . . . 94
A.4 L'utilisation des `axes` . . . 96
A.5 Collections et formes géométriques . . . 98
A.6 Légendes . . . 100

B Bases de données supplémentaires **103**
B.1 La base GEMS . . . 103
B.1.1 Requètes simples sur une table . . . 104
B.1.2 Requètes sur une ou plusieurs tables . . . 104
B.1.3 Les requètes emboitées . . . 104
B.2 Les Charts . . . 106
B.2.1 Requêtes simples . . . 106
B.2.2 Requêtes emboitées . . . 106

C LabelLines **107**

D Ressources internet **109**
D.1 SQL . . . 109
D.2 Python . . . 109
D.3 Basemap et SIG . . . 109

Table des figures

1.1 Schéma d'un carnet d'adresses . . . 14

3.1 L'observatoire de Saint-Maur en 2007 . . . 20

4.1 Schéma de la base Parcours . . . 27
4.2 Parcours : Schéma alternatif . . . 28
4.3 Différents types de jointures en SQL . . . 31

5.1 Nombre d'inscrits en Génie de l'Environnement depuis la création de la base `Parcours` . . . 36
5.2 Précipitations journalières à Saint-Maur des Fossés . . . 38
5.3 Température journalière, durant l'été, à l'observatoire de Saint-Maur . . . 43
5.4 Direction des vents à l'observatoire de Saint-Maur . . . 44

6.1 Tracé des côtes (projection Plate Carrée) . . . 50
6.2 Projection de Mollweide centrée sur le méridien de Paris. . . . 51
6.3 La France en Mercator transverse . . . 53
6.4 Deux façons de représenter l'échelle . . . 54
6.5 De Paris à Washington D.C. . . . 55
6.6 Sismicité mondiale (Février 2017) . . . 56
6.7 Variation temporelle de la composante verticale du champs magnétique $\dot{Z}$. . . . 57

7.1 Modèle de données OGR . . . 60
7.2 Pays du monde, projection Plate Carrée . . . 62
7.3 Population mondiale (en 2017). . . . 63

8.1 Landsat TM8 des landes d'Aquitaine. . . . 69
8.2 MNT ASTER de la Guadeloupe...en deux bouts . . . 70
8.3 MNT ASTER de la Guadeloupe...en un seul bout ! . . . 72
8.4 Bassin de Bayanbulak Landsat 5 144030, Canal 10. . . . 73
8.5 Histogramme des trois canaux (10,20,30) de la Landsat 144030 . . . 75
8.6 Bassin de Bayanbulak, Landsat 5 144030 ; image composée des trois canaux (R :30,V :20,B :10) étirés . . . 76
8.7 Bassin de Bayanbulak Landsat 5 145030, image composée des trois canaux (R :30,V :20,B :10) étirés . . . 76
8.8 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. . . . 80

9.1 Nombre agrégé d'étudiants par département un an avant leur arrivée à l'UFR STEP . . . 83
9.2 Hydrogéologie des bassins des étangs de Cazaux et de Parentis. Image satellite : ESRI World Imagery. Hydrogéologie : SIGES Aquitaine. . . . 85
9.3 Hydrogéologie des bassins des étangs de Cazaux et de Parentis 2ème version. . . . 88

A.1 $x\cos(x)$ . . . 95
A.2 Profils d'une nappe en état stationnaire pour différentes conductivités hydrauliques. . . . 96
A.3 Doubler un axe avec `axes`. . . . 97
A.4 Positionnement des repères avec `axes`. . . . 98
A.5 Utilisation des objets `collections` et `patches` . . . 100

B.1 Lien entre deux tables . . . 104
B.2 Schéma de la Base GEMS . . . 105

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 les données et créer des cartes en utilisant deux langages de programmation le `SQL` et `Python 2.7`. 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 le plus pratique possible.

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 est 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.

Première partie

Bases de données

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 (oups 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 "vraies" avec des colonnes de calculs, des graphiques et des commentaires. Tout se mélange. Le risque augmente alors que vous perdiez la connaissance de ce qui est et de ce qui n'est pas une donnée originale, parfois chèrement acquise.

Si vos données ont une valeur quelconque, elles doivent être stockées quelque part 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 année un langage de référence. notamment en sciences de la terre et en géomatique. Tous les grands logiciels de cartographie proposent des interfaces python.
— SQL (MySQL ou MariaDB) et Python sont indépendants des systèmes de vos ordinateurs

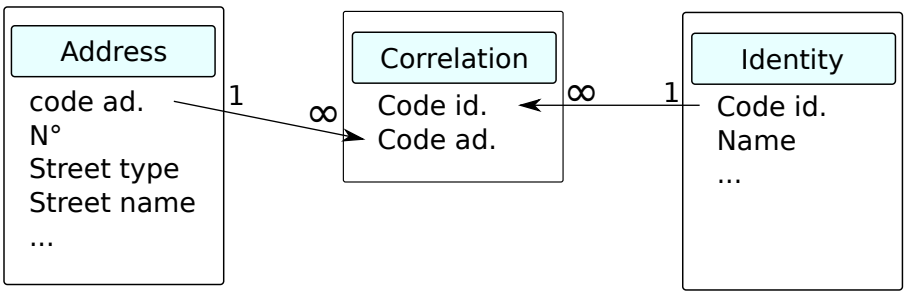


FIGURE 1.1 – Schéma d'un carnet d'adresses

— MariaDB et Python sont libres !

On pourrait ajouter que MySQL [1], 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 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 example 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 (peut être atteint) où. 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, permet de limiter les erreurs et les redondances et permet 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 un logiciel opensource, pour un équivalent opensource, vous pouvez installer MariaDB qui utilise la même syntaxe.

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

```
show databases;
```

NB : toute instruction SQL se finit par un « ;»
Pour se connecter à une base (test par example) on écrira

```
use test;
```

Pour créer la base `Toto` puis l'utiliser nous écrirons

```
create database Toto;
use Toto;
```

Pour détruire une base on écrira enfin

```
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

```
show tables;
```

Ici il n'y en a pas. Nous allons donc en créer une que nous appelerons `tata`. Contrairement à une base, lors de la création d'une table nous devons définir la table de façon détaillée. Nous devrons 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).

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

pour connaître la structure de `tata` on écrira

```
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

```
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 :

```
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

```
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` :

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

Enfin pour ajouter une colonne d'indexation automatique :

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

Pour effacer une colonne

```
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

```
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 desactivée 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

```
load data local infile "chemain/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

```
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

```
delete from tata where condition
```

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

```
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 example 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

```
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 ?

3
Les requètes sur un 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 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

```
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

```
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 :

```
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 ^oC
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 (^oC)
tairmin	float	minimum air temperature
tairmax	float	maximum air temperature
relhum	int	air relative humidity (%)
dewav	float	average dew point (^oC)
dewmin	float	minimum dew point
pairav	int	average air pressure ($mbar$)
pairmin	int	minimum air pressure
pairmax	int	maximum air pressure

TABLE 3.1 – imetos table structure

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

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

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

3.3.2 Ordonner

On peut ajouter un tri croissant à nos données de vent

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

Il est possible de trier sur deux champs, soit en augmentation ou en diminution.

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

l'argument `asc` est facultatif car c'est la valeur par défaut du classement. Si vous souhaitez un classement descendant vous devez l'indiquer avec `desc`.

3.3.3 Distinguer

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

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

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

```
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

```
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 miltiplicateur de 3.6 pour obtenir la vitesse en km/h

```
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

```
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.

```
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

```
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. La clause `distinct` fonctionne toujours.
5. En SQL, les comparaisons se font au moyen les 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 `like` et j'écrirai

```
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

```
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 %

```
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

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

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

```
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 shouhaite aussi savoir quand les mesures ont été prises. J'utiliserai pour ce faire une condition `where` avec une conjonction `and`

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

Maintenant, si je veux connaître la même chose mais avec une condition est moins stricte, par example je ne veux l'irradiation quand il n'y a pas de précipitations ou quand l'humidité est inférieure à 70%. Dans ce cas, la conjonction est `or`

```
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

```
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`

```
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) : arrondit 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 returne NULL.
- CONCAT_WS(SÉPARATEUR,STR1,STR2) : concaténation avec séparateur, ignore les NULL.
- LOCATE(SUBSTR,STR) : Position de la première occurence 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 prefixe 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 tire d'example, nous voulons calculer la pression de vapeur saturante [1] (e_a^*)

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

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

$$\frac{e_a}{e_a^*} = W_a \tag{3.2}$$

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

```
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.

1. Elle correspond à la pression de vapeur lorsque l'air est saturé et ne peut plus s'évaporer d'eau.

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 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

```
select date(date), sum(precip) /* Extract date (YYYY-MM-DD)
from the date field and calculate ∑_0^23 p */
from imetos
group by date(date); /* sets the boundaries of the sum
calculated above */
```

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

```
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`

```
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

```
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

```
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

```
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. Combien d'enregistrements dans la table imetos ?
2. Combien de valeurs distinctes de pression (moyenne, min et max) ?
3. Combien de valeurs distinctes de précipitations ?
4. Valeur maximale des précipitations ?
5. Vitesse maximale du vent ?
6. Pression minimale ?

7. Précipitations le 5 janvier ?
8. Précipitations et vitesse du vent en cas de pluie ?
9. Différence de température entre le sol et l'air le 10 janvier ?
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 emboitées

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

```
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

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

puis utiliser la valeur retournée dans une condition `where` :

```
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 :

```
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 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 peuvent également permettre de créer des sous-ensemble 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

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

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

```
show full tables;
```

ou

```
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

```
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.

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.

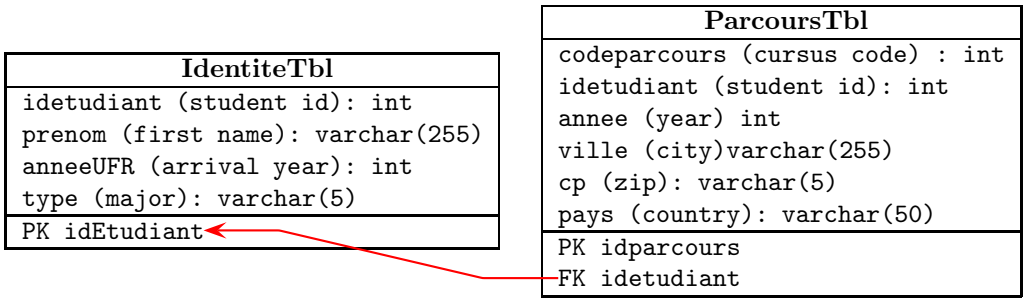


FIGURE 4.1 – Schéma de la base Parcours

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^o de sécurité sociale ou le N^o 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.

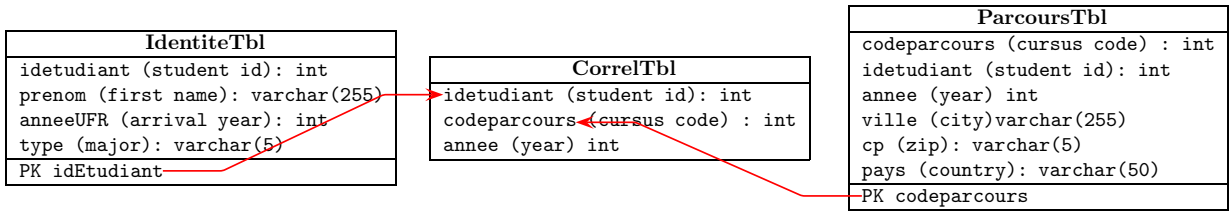


FIGURE 4.2 – Parcours : Schéma alternatif

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.

```
/* create database */
create database Parcours;

/* create table Identitetbl */
create table Identitetbl(
idetudiant int auto_increment primary key,
prenom varchar(255),
anneeUFR int,
type varchar(3)
);

/* create table Parcours */
create table ParcoursTbl(
codeparcours int auto_increment primary key,
idetudiant int not null,
annee int not null,
```

```
17 ville varchar(255),
18 cp varchar(5),
19 Pays varchar(255)
20 );
```

4.1.3 Insérer les données

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

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 :

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

2. pour effacer un enregistrement :

   ```
   delete from ParcoursTbl
   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/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 modèles/Patrons plus subtils utilisant la clause `regexp` (pour regular expression).

```
/* villes must contain p or a*/

select * from ParcoursTbl where ville regexp '[pa]'

/* villes must contain pa */

select * from ParcoursTbl where ville regexp 'pa'

/* villes must contain d or e or f */

select * from ParcoursTbl where ville regexp '[def]'
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

```
/* villes must start by p, a or c */
select * from ParcoursTbl where ville Like '^[p,a,c]'
/* villes must end by p, a or c */
select * from ParcoursTbl where ville Like '[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 étant que les enregistrements doivent avoir le même `idetudiant`.

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

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

Ici, le lien entre les deux tables est effectué via

```
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

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

ou bien encore

```
select IdentiteTbl.idetudiant, prenom, ville, cp, pays
from IdentiteTbl join ParcoursTbl
on  IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
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

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

ou

```
select IdentiteTbl.idetudiant, prenom, ville, cp, pays
from IdentiteTbl left join ParcoursTbl
on  IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
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

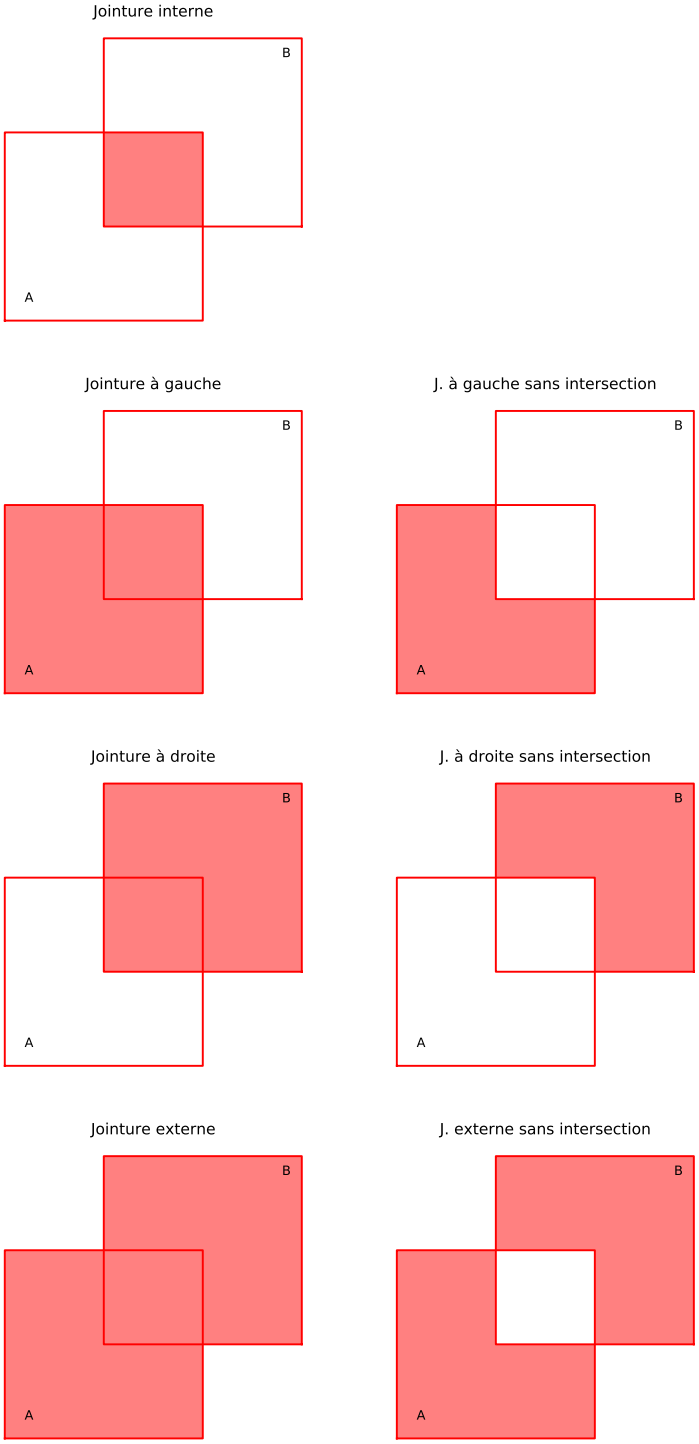
```
select IdentiteTbl.idetudiant, prenom, ville, cp, pays
from IdentiteTbl full join ParcoursTbl;
on  IdentiteTbl.idetudiant=ParcoursTbl.idetudiant
```

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



Jointure interne :

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

Jointure à gauche :

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

Jointure à gauche sans l'intersection de B :

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

Jointure à droite :

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
```

Jointure à droite sans l'intersection de A :

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

Jointure externe complète :

```
SELECT *
FROM A
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

```
(SELECT * FROM A LEFT JOIN B ON A.key = B.key)
UNION
(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...

Jointure externe sans intersection :

```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

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

Produit croisé

```
SELECT *
FROM A
CROSS JOIN B
```

4.2.4 Requêtes simples sur une ou deux tables

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 at-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 départment 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 emboitées

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 prénoment les étudiants qui ont étudié à Paris avant d'arriver à l'UFR ?

Pour faire disparaître le second charlotte utiliser la clause DISTINCT.

5
Utiliser Python et MySQL

5.1 Introduction

La connection à une base de données `MySQL` depuis `Python` se fait au moyen d'une librairie appellée `MySQLdb.py`. La procédure de connection est relativement simple et codifiée. Nous verrons les différentes étapes une à une. Tout se fait au moyen de classes (A.2.3), On créé 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 écrirons 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 Examples

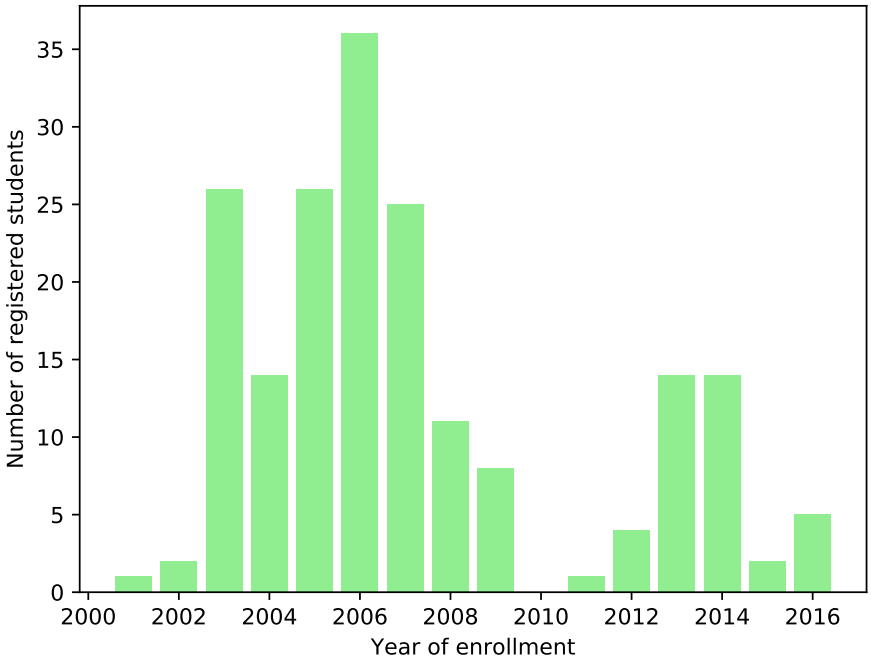
5.2.1 Récupérer des données numériques

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

```
# -*- coding: utf-8 -*-
"""
pm_1.py

First Python-Mysql script
"""
from pylab import *
import MySQLdb as msql

# 1) Create connection
conn=msql.connect(host="localhost",user="root",db="Parcours",passwd="root")

# 2) Create postman (cursor)
postman = conn.cursor()

# 3) write SQL GEI
sql="select anneeUFR, count(*) from IdentiteTbl where anneeUFR>1900 and type='GEI' group by
↪  anneeUFR"
```

```
18
19 # 4) Send letter
20 postman.execute(sql)
21
22 # 5) Receive answer
23 res=array(postman.fetchall())
24
25 # 6) Process
26 anneeGEI=res.transpose()[0]
27 inscritsGEI=res.transpose()[1]
28
29 # graphique
30 fig=figure()
31 bar(anneeGEI,inscritsGEI,color='lightgreen')
32 ylabel(u"Number of registered students")
33 xlabel(u"Year of enrollment")
34 #savefig("../figures/FigSQL1.pdf",bbox_inches='tight')
35 show()
36
37 # close the connexion
38 postman.close()
39 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
```

```
 3) plots the retrieved data
"""

# import statements
import MySQLdb # MySQL api
from matplotlib.pylab import *


# name of the database you whish to connect to
base='Parcelle'
# establish connection
conn=MySQLdb.connect(host="localhost",user="root",passwd="root",db=base)
# create a cursor object to send quaries
cursor=conn.cursor()
# write the SQL query
query='select date(date), sum(precip) from imetos group by date(date)'
# execute the query
cursor.execute(query)
# fetch the dataset
rows = cursor.fetchall()

# define the variables as an empty python list
da=[]
pr=[]
# for each record returned store the values
for row in rows:
        da.append(row[0])
        pr.append(row[1])

# plot the results
fig=figure(1)
bar(da,pr)
xlabel('Date')
ylabel('Precipitation in mm')

fig.autofmt_xdate() # command to get the dates plotted correctly
#savefig("../figures/pm_2.pdf",bbox_inches='tight')
show()

cursor.close()
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 utilisé 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`)

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

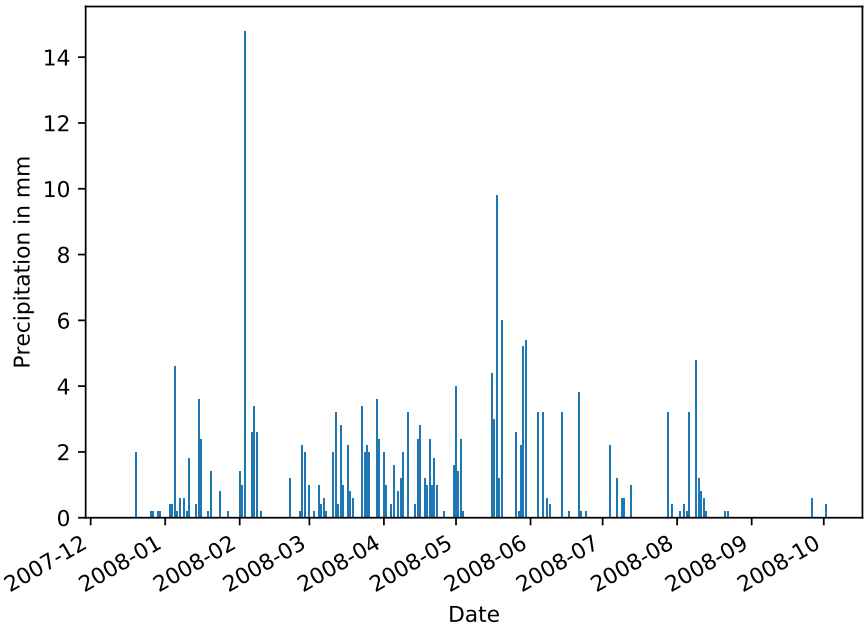


FIGURE 5.2 – Précipitations journalières à Saint-Maur des Fossés

```
pm_3.py

Python-Mysql DDL
"""
import MySQLdb

def create_connection(h=None,u=None,p=None):
        """Creates and returns a connection object"""
        Conn=None
        if h is None or u is None or p is None:
                print "You must valid host, user and passwords"
        else:
                try:
                        conn=MySQLdb.connect(host=h,user=u,passwd=p,
                        ↪  charset='utf8',use_unicode=True)
                except:
                        print "connection failed"
        return conn

def create_database(conn=None,dbname=None):
        """
        Establish connection to MySQL
        Create database dbname
        """
        if dbname is None or conn is None:
                print "You must provide a database name and valid connection"
                return
        else:
                # create base
                sql="create database if not exists %s;" % dbname

                cursor=conn.cursor()
                res=cursor.execute(sql)
                print res
                conn.commit() # Needed in the case when you write something in the database.
                ↪
```

```
                cursor.close()

def drop_database(conn=None,dbname=None):
    """
    Establish connection to MySQL
    drop database dbname
    """
    if dbname is None or conn is None:
        print "You must provide a database name and valid connection"
        return
    else:
                # drop base
        sql="drop database %s;" % dbname
        cursor=conn.cursor()
        res=cursor.execute(sql)
        print res
        conn.commit() # Needed in the case when you write something in the database.
        cursor.close()

def sql(conn=None,dbname=None,sqlstatement=None):
    """
    executes an sql statement (other than select) in database dbname
    """
    if dbname is None or conn is None or sqlstatement is None:
        print "You must provide a database name and valid connection and sql statement"
        return
    else:
        # create a cursor object to send quaries
        cursor=conn.cursor()
        # connect to the database
        cursor.execute("use %s;" % (dbname))
        conn.commit()
        # execute de query
        cursor.execute(sqlstatement)
        conn.commit()
        cursor.close()

    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 examples d'utilisation de ces fonctions :

```
# -*- coding: utf-8 -*-
"""
pm_3-1.py

Python-Mysql DDL
"""
from pm_3 import *

#creates connection
myconn=create_connection(h="localhost",u="root",p="root")

#creates database
create_database(myconn,"Toto")

#sql queries
#create table
sql(conn=myconn,dbname="Toto",sqlstatement="create table if not exists tata (val1 int, val2
↪ int, val3 float)")

```

```
#insert data
query="insert into tata values (1,2,3.0),(2,3,4.0),(3,4,5.0)"
sql(conn=myconn,dbname="Toto",sqlstatement=query)

# close connection
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 est préférable 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

```
req="SELECT * from mytable"
try:
        mycursor.execute(req)
except Exception as err:
        # send back the message
        msg="Incorrect SQL Query        : \n%s\nError detected:"%(req)
        return msg+str(err)
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.

```
#!/usr/local/bin/python
# coding: utf-8
# web, os and db
"""
openDB.py

Class OpenDB: wrapper to connect and execute SQL queries (DDL and DML)
"""
import operator, os, pickle, sys
import MySQLdb


class OpenDB:
    """
     class to encapsulate creation of connection and query execution to a Mysql Database
    """
    def __init__(self,base=None,user=None,passwd=None,host=None):
        if base is None or user is None or passwd is None or host is None:
            print "you must provide a valid database, host, user and password to create an
            ↪  OpenDB instance"
            return 1
        else:
            self.conn=MySQLdb.connect(host,user,passwd,base)

    def execQuery(self,req,param=()):
        cursor=self.conn.cursor()
        try:
            cursor.execute(req,param)
        except Exception as err:
            # send back the message
```

```
            msg="Incorrect SQL Query     : \n%s\nError detected:"%(req)
            return msg+str(err)
        res=[]
        if "SELECT" in req.upper():
            res= cursor.fetchall()
        else:
            self.conn.commit()
        cursor.close()
        return res

    def QueryToFile(self,req,fname='out.txt',param=()):
        """
        Executes a query and outputs to a text file for further processing (if needed)
        """
        res=self.execQuery(req,param)
        f=open(fname,'w')
        for r in res:
            for i in npy.arange(len(r)-1):
                st="%s\t" % (r[i])
                f.write(st)
            st="%s\n" % (r[len(r)-1])
            f.write(st)

        f.close()

    def close(self):
        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

```
# -*- coding: utf-8 -*-
"""
pm_4.py

same as pm_2.py but the data retrieval uses the OpenDB class
"""

# import statements
import MySQLdb # MySQL api
from matplotlib.pylab import *
from openDB import *


#get the data
#establish the connection
o=OpenDB(base="Parcelle",host="localhost",user="root",passwd="root")
#write and execute the query
sql="select date(date), sum(precip)  from imetos group by date(date)"
rows=o.execQuery(sql)

# define the variables as an empty python list
da=[]
pr=[]
# for each record returned store the values
for row in rows:
        da.append(row[0])
        pr.append(row[1])

```

```
# plot the results
fig=figure(1)
bar(da,pr)
xlabel('Date')
ylabel('Precipitation in mm')

fig.autofmt_xdate() # command to get the dates plotted correctly
show()

#close database
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

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

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

```
# -*- coding: utf-8 -*-
"""
pm_5.py

Plot of the hourly temperatures at Saint-Maur averaged over july,August and september
"""
#import libraries
import matplotlib
matplotlib.use('GTKAgg')
from matplotlib.pylab import *
from openDB import *

#get the data
#establish the connection
o=OpenDB(base="Parcelle",user="root",passwd="root",host="localhost")
#write and execute the query
sql="select hour(date), avg(tairav) from imetos  where month(date) in (7,8,9) group by
 ↪  hour(date)"
res=array(o.execQuery(sql))
o.close()

#retrieve the results
h=res.transpose()[0]
ta=res.transpose()[1]

#plot the result
fig=figure()
plot(h,ta,'ro-')

xlabel("Hour of the day",fontsize=14)
ylabel('Average temperature $^oC$',fontsize=14)
axis([0, 24, 10, 25 ])
grid(True)

show()
```

La figure 5.3 montre le résultat.

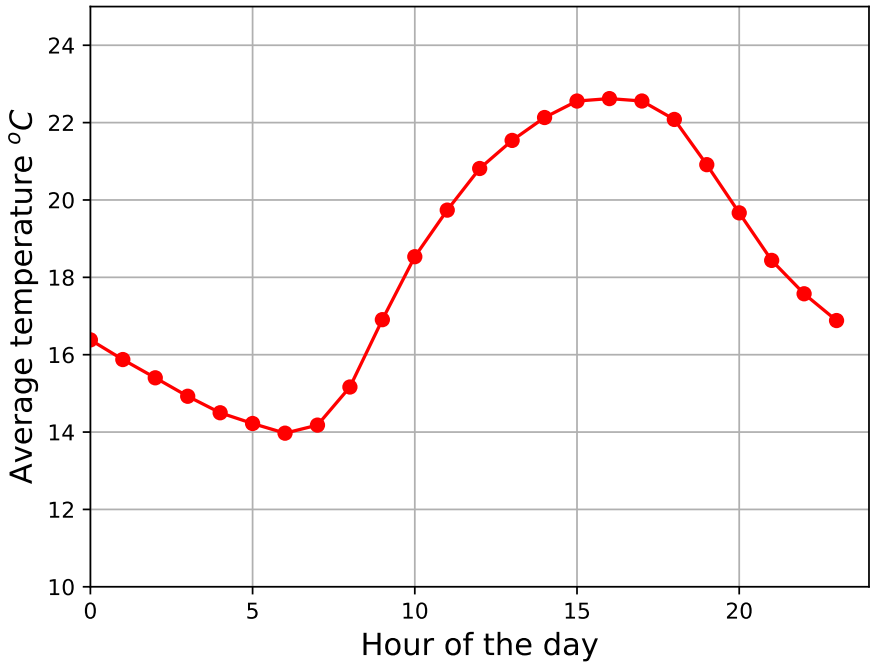


FIGURE 5.3 – Température journalière, durant l'été, à l'observatoire de Saint-Maur

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. Un requête `SQL` permet de répondre à cette question. Il s'agit de

```
select round(winddir/20)*20, count(winddir) from imetos
group by round(winddir/20)*20
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.

```
# -*- coding: utf-8 -*-
"""
pm_6.py

plots the histogram of wind directions at Saint-Maur as a polar plot.
"""
#import libraries
from matplotlib.pylab import *
from openDB import *


#get the data
#establish the connection
o=OpenDB(base="Parcelle",user="root",passwd="root",host="localhost")
#write and execute the query
sql="select round(winddir/20)*20, count(winddir) from imetos group by round(winddir/20)*20
 ↪  order by winddir"

res=o.execQuery(sql)

print res
#retrieve the results
radii=[]
theta=[]
for r in res:
```

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

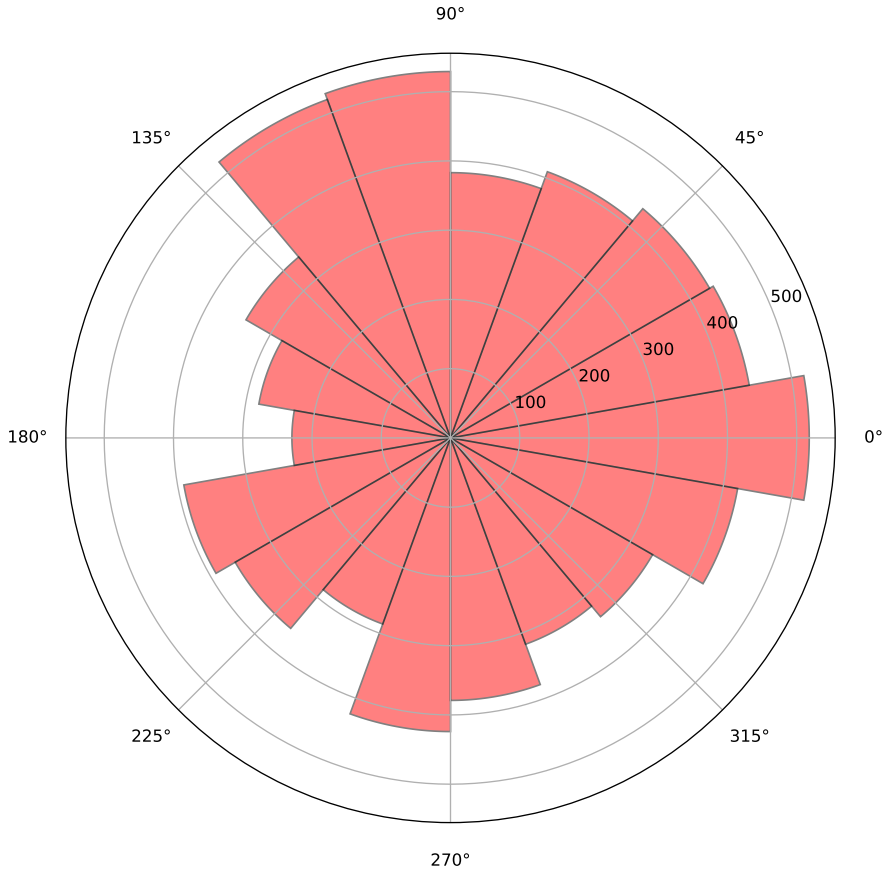


Figure 5.4 – Direction des vents à l'observatoire de Saint-Maur

5.5 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 notamment des tables simples comme la table `imetos` de `Parcelle`.

```
"""
pm_7.py

use pandas to read sql data from Mysql database
"""
#library needed to create a connection that pandas can use
from sqlalchemy import create_engine
# pandas
import pandas

# default syntax engine = create_engine('mysql://user:passwd@host/dbname')
engine = create_engine('mysql://root:root@localhost/Parcelle')

df=pandas.read_sql_table('imetos',engine)
print df.head()
```

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 applicant à imetos.

```
"""
pm_8.py

use pandas to read sql data from Mysql database and compare syntaxes
"""
#library needed to create a connection that pandas can use
from sqlalchemy import create_engine
# pandas
import pandas

# default syntax engine = create_engine('mysql://user:passwd@host/dbname')
engine = create_engine('mysql://root:root@localhost/Parcelle')

df=pandas.read_sql_table('imetos',engine)
print df.head()


print("Number of hours of precipitation")
# select count(*) from imetos where precip > 0
print df[df["precip"]>0]["date"].count()


print("Number of hours of precipitation during night time")
# select count(*) from imetos where precip > 0 and solrad=0
print df[(df["precip"]>0) & (df["solrad"]==0)]["date"].count()


print("Monthly Averaged hourly precipitation  during daytime")
# select month(date), avg(precip) from imetos where solrad >0 group by month(date)
print df[df["solrad"]>0].groupby(df["date"].dt.month)["precip"].mean()

#etc...
```

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 crées à l'aide de variables aléatoires de distribution uniforme.

```
"""
pm_9.py

use pandas to write pandas data to a MySQL table
"""
#library needed to create a connection that pandas can use
from sqlalchemy import create_engine
# pandas
import pandas
import numpy as np

# connect to the Toto db
engine = create_engine('mysql://root:root@localhost/Toto')

# create a dataframe with three series val1, val2 val3 of 100 000 values
# val 1 in [0,10]
val1 = np.random.randint(0,10,100000)
print val1
# val 2 in [0,100]
val2 = np.random.randint(0,100,100000)
print val2
# val 3 in [0,1000.0] uniform distribution
val3 = np.random.rand(100000)*1000.0
print val3

# create the frame
df=pandas.DataFrame({"val1": val1.tolist(), "val2":val2.tolist(),"val3":val3.tolist()})
print df.head()

#store the data in table tata
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.

Deuxième partie

Cartographie

6
La représentation cartographique avec basemap

6.1 Introduction

basemap est une extension de la librairie graphique matplotlib qui permet de faire des cartes. Chaque carte produite par l'intermédiaire de basemap est une figure de matplotlib. Ceci présente l'immense 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'interfacer avec les systèmes de gestion de données comme MariaDB (MySQL) que nous utilisons en première partie. 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].

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

1. Le tutoriel de basemap consultable en ligne [2]. Étant donné que vous avez déjà eu un cours de python nous irons plus vite que le tutoriel.
2. Divers tutoriels portant sur la librairie GDAL [3] qui est au coeur de toutes les applications de SIG gratuites ou payantes aujourd'hui.

Librairies nécessaires : basemap, pyshp, pyproj, shapely, geopandas. Tous les tutoriels en ligne expliquent comment installer ces librairies, je n'insiste donc pas.

Enfin signalons ici que basemap évolue progressivement vers une bibliothèque nommée cartopy [4]. Pour l'instant le portage reste partiel et l'installation demande la compilation des sources. Je continue donc d'utiliser basemap d'autant que le passage d'une librairie à l'autre est assez simple.

6.2 Première carte avec basemap

Nous utilisons dans cette première section des données intégrées à la librairie.

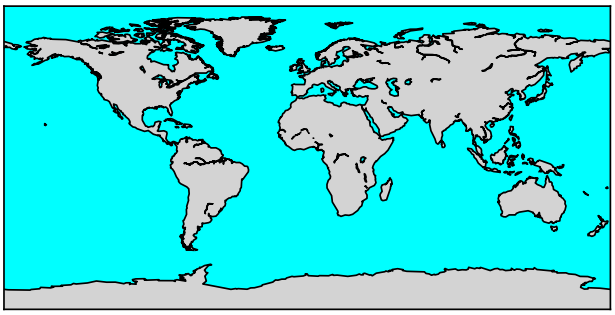
```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_1.py

simple first map (taken from basemap tutorial)
"""

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

#Define the map object
```

1. Message du professeur Gayer : ≪le shapefile est au Raster ce qu'Illustratior (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://basemaptutorial.readthedocs.io/en/latest/index.html
3. https://pcjericks.github.io/py-gdalogr-cookbook/
4. https://scitools.org.uk/cartopy/docs/v0.13/index.html

```
13 map = Basemap()
14
15
16 #Fill the globe with a blue color
17 map.drawmapboundary(fill_color='aqua')
18
19 #Fill the continents with the land color
20 map.fillcontinents(color='lightgray',lake_color='aqua')
21
22 #draw the coastlines
23 map.drawcoastlines()
24
25 #plt.savefig("../figures/bm1.pdf",bbox_inches="tight")
26 plt.show()
```

Ce qui donne la figure 6.1.

FIGURE 6.1 – Tracé des côtes (projection Plate Carrée)



6.3 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'occurence).

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)ous

Comme la terre n'est pas totalement sphérique on utilise un ellipsoide afin d'améliorer le positionnement précis d'un point sur le globe. Le globe n'étant pas non plus totalement ellispoidal des variations locales de forme de l'ellipsoide 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 example)

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 site, 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 et leur système de référence.

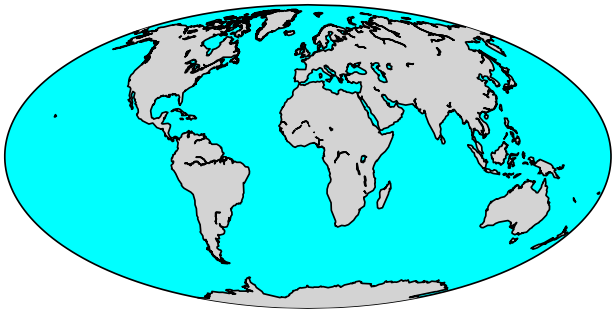
6.3.1 Sélectionner une projection dans basemap

Dans `basemap` le choix de la projection est un argument de la déclaration de l'objet map. Suivant la projection des arguments devront être insérés. La projection par défaut est la Plate carrée ou projection cylindrique équidistante. La nature et les propriétés de la projection que l'on souhaite sont passées comme argument de l'appel `map.basemap(**args)`. Pour une projection de Mollweide on devra fournir la longitude centrale soit `map`

= Basemap(projection='moll', lon_0=0). Les informations concernant chacune des projections acceptée par basemap se trouve dans l'aide en ligne[5]

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_2.py

Change basemap projection
"""
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

map = Basemap(projection='moll', lon_0 = 0)

map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='lightgray',lake_color='aqua')
map.drawcoastlines()

plt.show()
```

FIGURE 6.2 – Projection de Mollweide centrée sur le méridien de Paris.



Limitation : si vous voulez représenter des données vous devez connaître la projection dans laquelle celles-ci sont enregistrées avant de créer l'instance de basemap. Il n'est pas possible de changer la projection une fois la carte crée. On peut récupérer les informations de projections dans un format standard (PROJ.4) et s'en servir pour faire des conversion. Si vous voulez que votre carte soit dans la projection de vos données il faut commencer par trouver celle-ci.

6.3.2 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 example de la façon dont pyproj permet de changer de coordonnées et de calculer des distances.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_3.py

Coordinate transormations usis pyproj
"""
#libray
import pyproj

#
# Define  original coordinates
```

5. https://matplotlib.org/basemap/users/mapsetup.html

```
#
UTM_X=565718.
UTM_Y=3980998.

#
# Define source and destination projections
#
srcProj=pyproj.Proj(proj="utm",zone="11",ellps="clrk66",units="m")
dstProj=pyproj.Proj(proj="longlat",ellps="WGS84",datum="WGS84")

#
# Do the  corrdinate transformation
#
Long,Lat=pyproj.transform(srcProj,dstProj,UTM_X,UTM_Y)
print "Coordinate transformation from UTM 11 zone \n\t %0.4f,%0.4f => %0.4f E,%0.4f N" %
 ↪  (UTM_X,UTM_Y,Long,Lat)

#
# Add ten kilometers to one UTM coordinate
#
UTM_Y+=10000
Long2,Lat2=pyproj.transform(srcProj,dstProj,UTM_X,UTM_Y)
print "Coordinate transformation from UTM 11 zone\n\t %0.4f,%0.4f => %0.4f E,%0.4f
 ↪  N"        % (UTM_X,UTM_Y,Long2,Lat2)

#
# Find de position of a point located 10km north of the original point
#
azimuth=360
distance=10000

geod=pyproj.Geod(ellps="WGS84") # instantiate the Geod class to do the calculation
Long3,Lat3,invangle=geod.fwd(Long,Lat,azimuth,distance)
print "The point located 10km to the North of  %0.4f E,%0.4f N has  longitude  %0.4f E and
 ↪  latitude %0.4f 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.

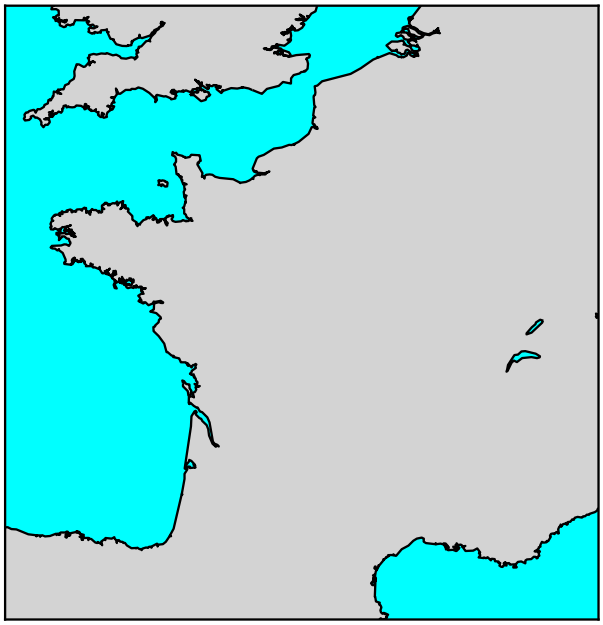
6.4 Les limites de la carte

Tout est dans la ligne de définition de la carte. On commence par donner les limites en long et lat du coin bas gauche (`llcrnrlon` et `llcrnrlat`) puis celles du coin droit haut (`urcrnrlon`, `urcrnrlat`). suit la projection (tmerc pour UTM) et les coordonnées du centre de la projection.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_4.py

Map limits
"""
```

```
8
9  from mpl_toolkits.basemap import Basemap
10 import matplotlib.pyplot as plt
11
12 # everything is in this line !
13 map = Basemap(llcrnrlon=-5,llcrnrlat=42,urcrnrlon=9.,urcrnrlat=52.,
14              resolution='i', projection='tmerc', lat_0 = 46, lon_0 = 2.5)
15
16 map.drawmapboundary(fill_color='aqua')
17 map.fillcontinents(color='lightgray',lake_color='aqua')
18 map.drawcoastlines()
19
20 plt.show()
```

FIGURE 6.3 – La France en Mercator transverse



Que constatez-vous si vous remplacez `i` par `f` dans l'argument `resolution` ? Notez que certaines projection n'acceptent pas le découpage comme la projection de Mollweide (`moll`) par exemple qui est faite pour représenter un globe entier.

Quand cela est possible il existe trois façons de délimiter un espace cartographique dans basemap :

1. celle que nous venons de voir qui consiste à entrer les coordonnées des limites en lat/lon ;
2. on peut aussi entrer les coordonnées dans le reprère de la carte (en x,y) en utilisant les arguments `llcrnrx,llcrnry,urcrnrx,urcrnry` ;
3. on peut enfon entre une largeur, une hauteur et un centre (`width,height,lon_0,lat_0`. À noter que l'ordonnée du coin gauche inférieur est (0,0) et celle du coin droit supérieur (`width,height`)

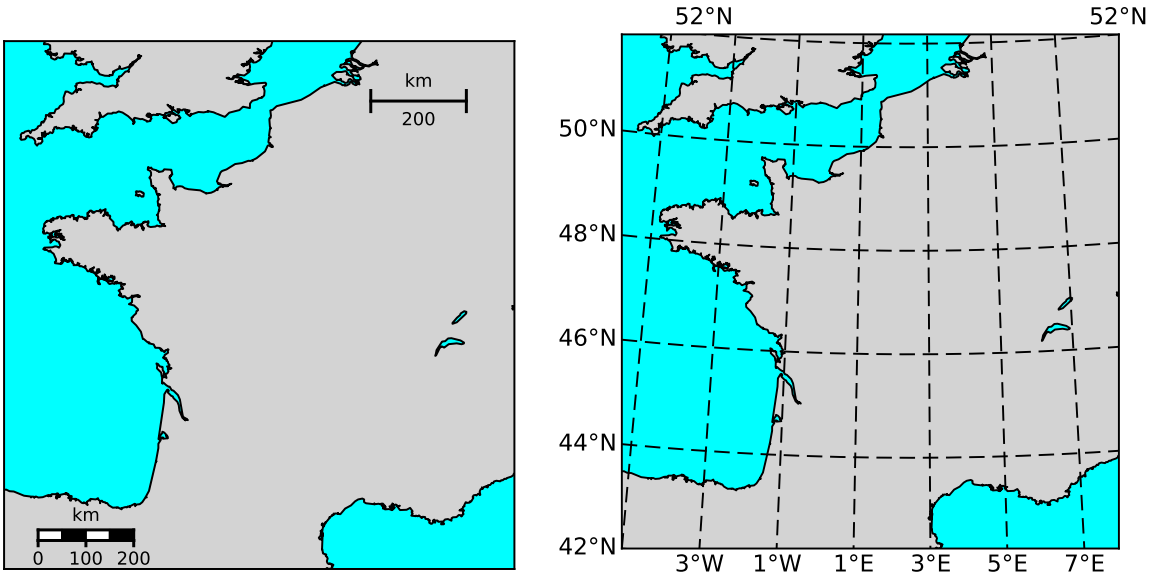
Le tutoriel de basemap explique tout cela très bien.

6.5 Une échelle svp

Pas de carte sans échelle s'il vous plaît ! Vous avez deux solutions : (1) une barre d'échelle en bonne et due forme ou (2) ce que l'on appelle un graticule à savoir une grille faite de méridiens et de parallèles. La première solution permet de se faire une idée des distances en un coup d'oeil. Elle n'est valable que dans le cas d'une carte suffisamment petite ou bien quand la projection conserve les distances. La seconde solution est valable tout le temps. Elle permet en outre de se faire une idée de la déformation de la carte. Il est plus difficile cependant d'appréhender les distances rapidement.

On reprend donc le script précédent et on ajoute

FIGURE 6.4 – Deux façons de représenter l'échelle




```
# first lon/lat = location of the bar
# second lon/lat = location at which distance is calculated
map.drawmapscale(-3., 42.75, 1, 47, 200, barstyle='fancy')
map.drawmapscale(6,51, 1, 47, 200)
```

dans le premier cas et

```
map.drawmeridians(range(-5,10,2),color='k',dashes=[4, 2], labels=[0,0,0,1],fontsize=12)
map.drawparallels(range(40,54,2),color='k',dashes=[4, 2],labels=[1,0,1,0],fontsize=12)
```

dans le second, ce qui donne respectivement les deux cartes de la figure 6.4. Il est intéressant de noter que, dans le premier cas, on calcule la distance à une position donnée sur la carte. Ceci montre bien l'intérêt de tracer une grille dès que l'emprise de la carte est importante.

6.6 Représenter des données

La commande `map=Basemap(...)` crée un objet carte et la figure correspondante. Pour représenter des données on peut donc avoir recours à des fonctions propres à l'objet map ou à matplotlib.

6.6.1 Trajet entre deux deux capitales

le script suivant montre comment représenter deux points, un pour Paris et un pour Washington D.C, comment nommer les deux villes et enfin tracer un grand cercle reliant ces deux capitales (figure 6.5).

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_5.py

Draw points and great circle with scale
"""

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

#define the figure size
fig=plt.figure(figsize=(20,10))
```

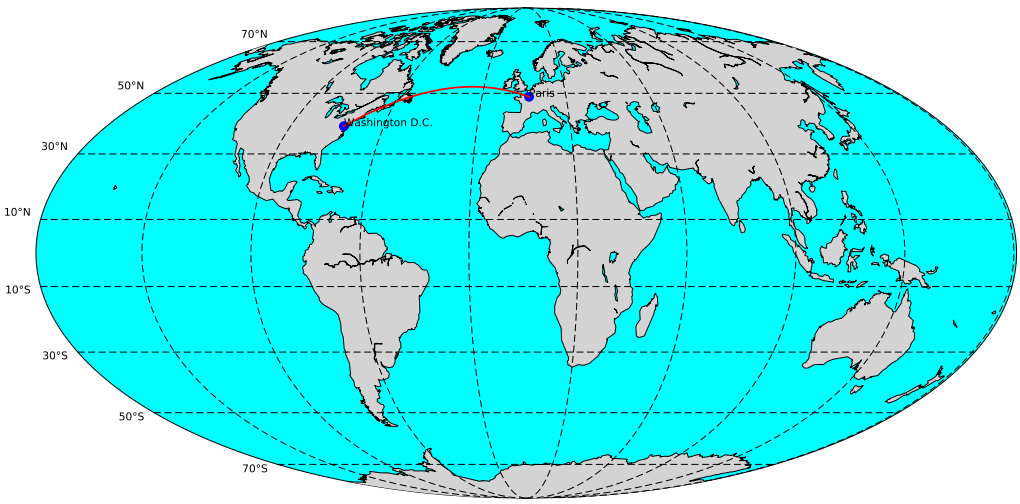
```
14
15 #define the basic map
16 map = Basemap(projection='moll', lon_0 = 1)
17 map.drawmapboundary(fill_color='aqua')
18 map.fillcontinents(color='lightgray',lake_color='aqua')
19 map.drawcoastlines()
20
21
22 #two samples of plots using map.plot and plt.plot
23 x,y=map(2.4, 48.8)
24 map.plot(x,y,'bo', ms=10)
25 offset=200
26 # text() is only a method of plot
27 plt.text(x+offset,y+offset,"Paris",fontsize=12)
28
29 x,y=map(-77.15, 38.89)
30 plt.plot(x,y,'bo', ms=10)
31 plt.text(x+offset,y+offset,"Washington D.C.",fontsize=12)
32
33 #draw a great circle linking the two cities
34 map.drawgreatcircle(2.4, 48.8, -77.15, 38.89,linewidth=2,color='r')
35
36 # and the scale
37 map.drawmeridians(range(-180,180,40),color='k',dashes=[4, 2])
38 map.drawparallels(range(-90,90,20),color='k',dashes=[4,
   ↪ 2],labels=[1,0,1,0],fontsize=12,zorder=1)
39
40 plt.show()
```

On notera

1. que la méthode (fonction) `plot()` est commune à `map` et à `plt` ;
2. que la méthode `text()` est propre à `plt` ;
3. que la méthode `drawgreatcircle()` est propre à `map`.
4. que la fonction `map(lon,lat)` renvoie les coordonnées x,y d'un point dans le système de la carte (x et y peuvent être des listes).

C'est l'immense force de `basemap` que de pouvoir faire appel aux fonctions de `matplotlib`.

FIGURE 6.5 – De Paris à Washington D.C.

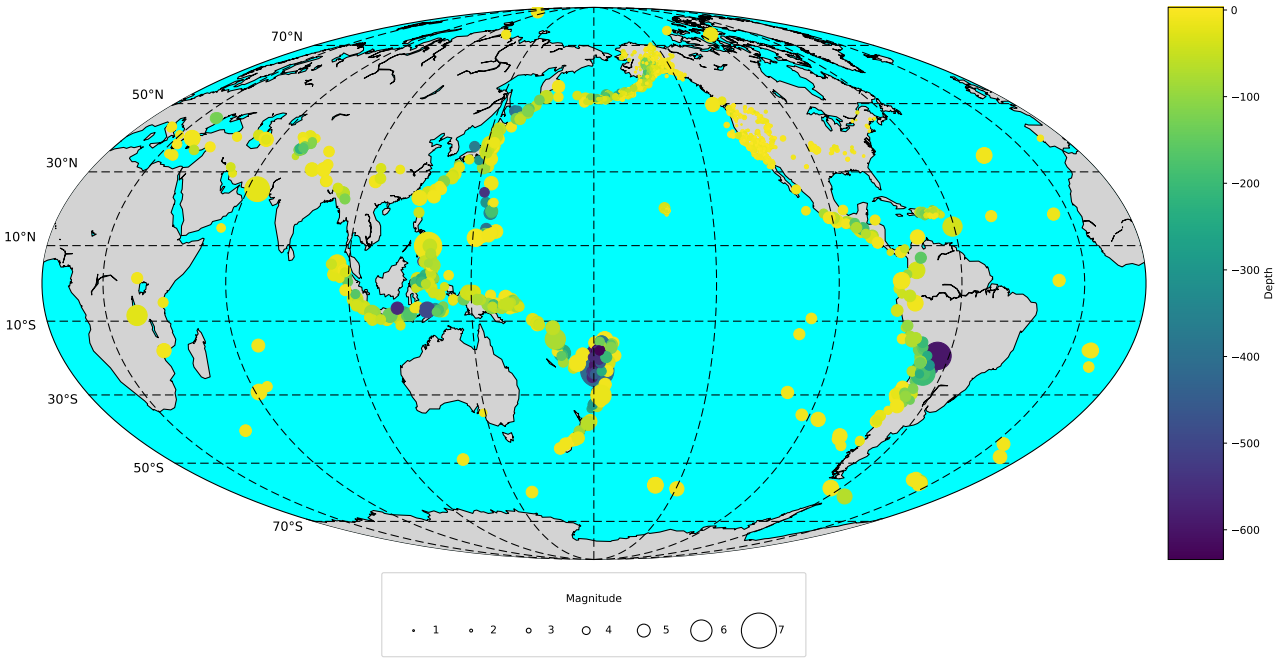


6.6.2 Carte de la sismicité (Scatter plot)

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 6.6 (aux légendes près).

FIGURE 6.6 – Sismicité mondiale (Février 2017)



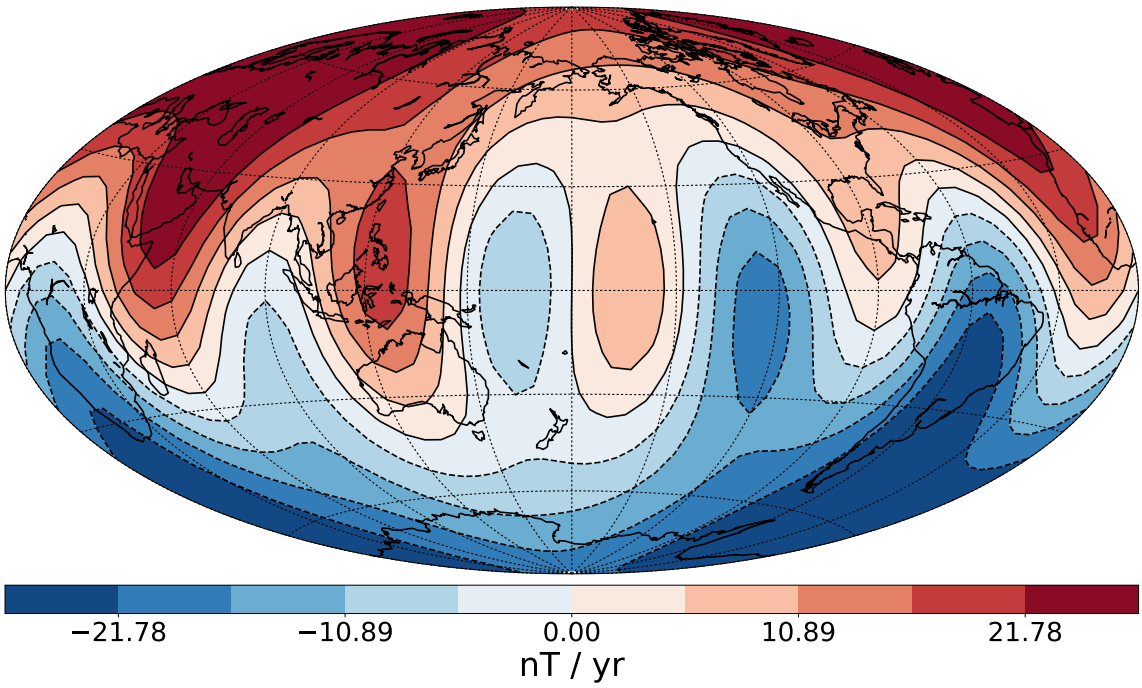

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*
"""
bm_5_EQ.py

Draw map of Feb 2017 Earthquakes
color=depth
size=magnitude

Original version and data  by E. Gayer
"""

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from matplotlib.patches import Circle
from matplotlib.patches import Patch
from matplotlib.collections import PatchCollection

#Figure
fig=plt.figure(figsize=(20,10))
ax=fig.add_subplot(111)

#Basic map
map = Basemap(projection='moll', lon_0 = 180)
map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='lightgray',lake_color='aqua')
map.drawcoastlines()

EQall_month=pd.read_csv('../csv/EQall_month.csv')
print EQall_month['time']

print min(EQall_month['mag']), max(EQall_month['mag'])
```

```
x,y=map(EQall_month['longitude'].tolist(), EQall_month['latitude'].tolist())
map.scatter(x,y,s=np.exp(EQall_month['mag']), c=-EQall_month['depth'], cmap='viridis',
↪  zorder=2)

#Legends
#color bar
cbar=map.colorbar()
cbar.set_label('Depth')

# Tricky legend for magnitude
for i in range(7):
    plt.scatter([], [], s=np.exp(i+1), edgecolor='k',facecolor='none',label=str(i+1))

h, l = plt.gca().get_legend_handles_labels()
ax.legend(h, l,title="Magnitude",labelspacing=2,borderpad=2,frameon=True, ncol=7,
↪  framealpha=0.9,bbox_to_anchor=(0.5,-0.1),loc='center')

# Scale
map.drawmeridians(range(-180,180,40),color='k',dashes=[4, 2])
map.drawparallels(range(-90,90,20),color='k',dashes=[4,
↪  2],labels=[1,0,1,0],fontsize=12,zorder=1)

# See the result
plt.show()
```

6.6.3 Variation temporelle du champs 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 champs magnétique $\mathbf{B}$ [6]. 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 6.7

FIGURE 6.7 – Variation temporelle de la composante verticale du champs magnétique $\dot{Z}$.




```
#!/usr/bin/env python2
# -*- coding: utf-8 -*
```

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

```
"""
bm_5_SVZ.py

Draw map of the time variation of the Z component of the magnetic field
Courtesy of A. Fournier
"""

import numpy as np
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

SVZ=np.loadtxt('../csv/SVZ.txt')
#size of the grid over the sphere
nlon = 61 # 61 points in longitude
nlat = 31 # 31 points in colatitude
#coordinates of SVZ points
colat = np.linspace(1., 179., nlat, endpoint=True) # we exclude the poles
longi = np.linspace(0., 360., nlon, endpoint=True)


plt.rc('legend', fontsize=30)
plt.rc('axes', labelsize=30,facecolor='None')
plt.rc('savefig', facecolor='None', edgecolor='None')

fig = plt.figure(figsize=(20,10))
map = Basemap(projection='hammer', lon_0=180)
map.drawcoastlines(linewidth = 1.5)
#   draw lat/lon grid lines every 30 degrees.
map.drawmeridians(np.arange(0, 360, 30))
map.drawparallels(np.arange(-90, 90, 30))

#transform coordinates and reshape SVZ into a 2D grid
latit = 90 - colat
lons, lats = np.meshgrid(longi,latit)
x, y = map(lons, lats)
svz2d =  np.reshape(SVZ, (nlat,nlon) )

#define color levels
maxsvz = max(np.abs(SVZ))
clevs = np.linspace(-maxsvz, maxsvz, 11, endpoint=True)
#plot SVZ
cs1 = map.contour(x, y, svz2d, clevs, colors='k', linewidth=0.5)
cs2 = map.contourf(x, y, svz2d, clevs, cmap=plt.cm.RdBu_r)
#add colorbar
cb = map.colorbar(cs2,"bottom", size="5%", pad="2%")
font_size = 24 # Adjust as appropriate.
cb.ax.tick_params(labelsize=font_size)
cb.set_label('nT / yr')

plt.show()
```

7
Les shapefiles

7.1 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/caracéristique (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

7.2 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 OGR [1]. 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 7.1).

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.

7.3 Shapefile structure

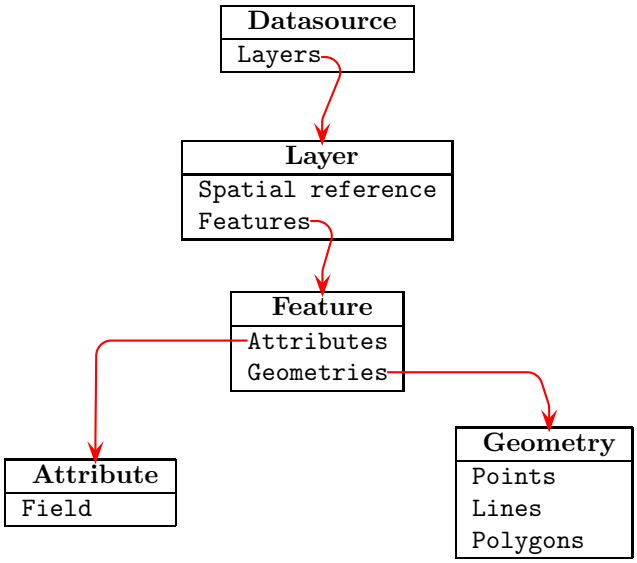
Les types de fichiers les plus utilisés pour stocker des données vectorielles sont ce que l'on appelle des fichier de formes (shapefile, extension .shp). Leur structure suit le standard OGR décrit en figure 7.1. Il est donc nécessaire de comprendre comment sont organisées les données à l'intérieur d'un 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.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_6.py

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 7.1 – Modèle de données OGR




```
import osgeo.ogr

# open shapefile
shapefile=osgeo.ogr.Open("./shapefiles/ne_50m_admin_0_countries/ne_50m_admin_0_countries.shp")

numLayers=shapefile.GetLayerCount() # counts layers
print "Shapefile contains %d layers" % (numLayers)
print

# Gets the reference system and the number of features for each layer
# In the case of GEOFLA Each layer has a spatial reference ! this complicates everything !!!
for LayerNum in range(numLayers):
    Layer=shapefile.GetLayer(LayerNum)
    numFeatures=Layer.GetFeatureCount()
    print "Layer %d has %d features" % (LayerNum,numFeatures)
    # tests whether there is a spatial ref attached to the file
    if Layer.GetSpatialRef():
        spatialRef=Layer.GetSpatialRef().ExportToProj4()
        print "Layer %d has spatial reference %s" % (LayerNum,spatialRef)
    else:
        print "Layer %d has no spatial reference" % (LayerNum)

    # gets the number of attributes per feature
    for featureNum in range(numFeatures):
        feature=Layer.GetFeature(featureNum)
        fieldcount=feature.GetFieldCount()
        print "Feature %d has %d attributes\n" % (featureNum,fieldcount)

    # prints the attributes of the last feature
    print "Feature %d has the following attributes:" % (featureNum)
    attributes=feature.items()
    for key,value in attributes.items():
        print "%s =%s" % (key,value)

    # retrieve the feature geometry and print its type
    geometry=feature.GetGeometryRef()
```

```
    # geometries are stored in map units not in lat/lon this also complicates things for basemap.
    print geometry.Centroid().ExportToWkt()
    geometryName=geometry.GetGeometryName()

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. Dans ce cas les choses se compliquent pour `basemap` car on devra passer par une étape de conversion des données et de recherche de la projection.

7.4 Représenter un shapefile

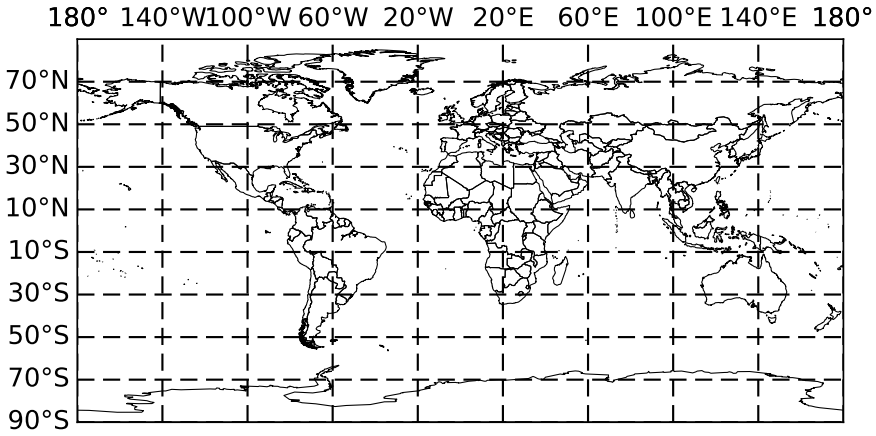
7.4.1 Contours

La représentation des géometries associées aux objets d'un fichier shape se fait par l'intermédiaire de la méthode `readshapefile()` de `basemap`.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_7.py

Read a shapefile
"""
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

#map definition
map = Basemap()

#read shapefile and plot
map.readshapefile('./shapefiles/ne_50m_admin_0_countries/ne_50m_admin_0_countries','wc')

#scale
map.drawmeridians(range(-180,180,40),color='k',dashes=[4,2],labels=[0,0,1,0],fontsize=12,zorder=1)
map.drawparallels(range(-90,90,20),color='k',dashes=[4,2],labels=[1,0,0,0],fontsize=12,zorder=1)

plt.show()
```

Commentaires. `readshapefile` a deux arguments obligatoires : le nom du fichier sans son extension, un nom d'objet qui servira a appeler les données du fichier chargé.

Figure 7.2 – Pays du monde, projection Plate Carrée



7.4.2 Colorier les polygones

`readshapefile()` permet de lire et charger un fichier shape et de représenter les géométries associées. Ces capacités graphiques sont cependant très limitées. Pour colorier des polygones par example il nous faudra

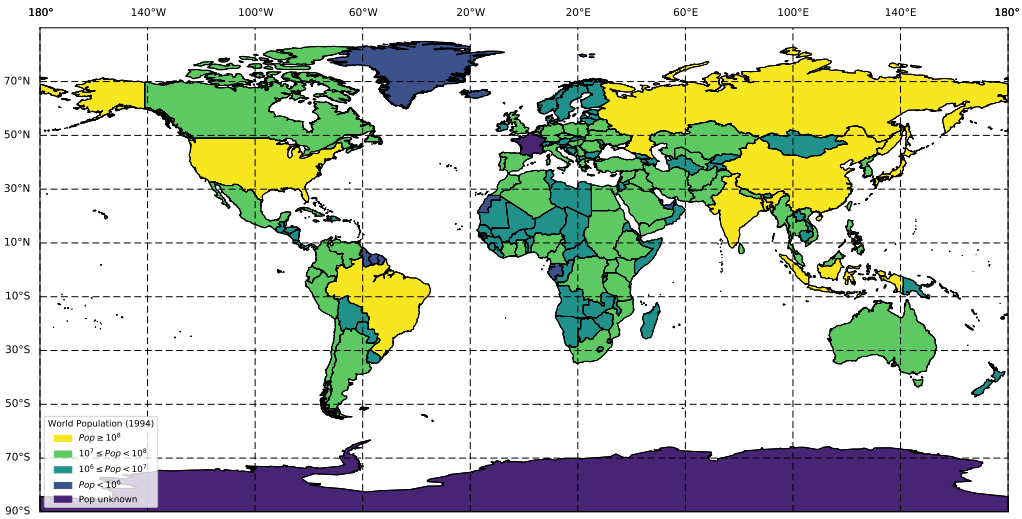
1. Récupérer les géométries de chaque objet ;
2. Les colorer en fonction des valeurs des attributs de l'objets.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_8.py

Fill polygons with colours
"""

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from matplotlib.patches import Patch
from matplotlib.collections import PatchCollection
from matplotlib.patches import PathPatch
import numpy as np
from matplotlib import cm


# definition of the figure and axis
fig     = plt.figure(figsize=(20,10))
ax      = fig.add_subplot(111)

#color map I use viridis because it is nice !
cmap=cm.get_cmap('viridis')

#map definition
map = Basemap()

#read shapefile and plot
map.readshapefile('./shapefiles/ne_50m_admin_0_countries/ne_50m_admin_0_countries','wc')


# filling polygons
p1,p2, p3,p4,p5   = [],[],[],[],[]

#selects and colors the countries that have more then 100M inhabitants
```

```
for info, shape in zip(map.wc_info, map.wc):
    if info['POP_EST'] >= 1e8:
        x,y=zip(*shape)
        p1.append( Polygon(zip(x,y), closed=True))
    elif info['POP_EST'] >= 1e7 and info['POP_EST'] < 1e8:
        x,y=zip(*shape)
        p2.append( Polygon(zip(x,y), closed=True))
    elif info['POP_EST'] >= 1e6 and info['POP_EST'] < 1e7:
        x,y=zip(*shape)
        p3.append( Polygon(zip(x,y), closed=True))
    elif info['POP_EST'] >= 0 and info['POP_EST'] < 1e6:
        x,y=zip(*shape)
        p4.append( Polygon(zip(x,y), closed=True))

#making the collections
pp1=PatchCollection(p1, facecolor= cmap(0.99) ,edgecolor='k')
ax.add_collection(pp1)
pp2=PatchCollection(p2, facecolor= cmap(0.75) ,edgecolor='k')
ax.add_collection(pp2)
pp3=PatchCollection(p3, facecolor= cmap(0.5),edgecolor='k')
ax.add_collection(pp3)
pp4=PatchCollection(p4, facecolor= cmap(0.25),edgecolor='k')
ax.add_collection(pp4)
pp5=PatchCollection(p5, facecolor= cmap(0.1),edgecolor='k')
ax.add_collection(pp5)

#and the legend
colors = [cmap(0.99), cmap(0.75), cmap(0.5), cmap(0.25),cmap(0.1)]
texts = ["$Pop \geq 10^8$", "$10^7 \leq Pop < 10^8$","$10^6 \leq Pop < 10^7$", "$Pop <
↪  10^6$","Pop unknown"]
patches = [ Patch(color=colors[i], label=texts[i] ) for i in range(len(texts)) ]
plt.legend(handles=patches, loc='lower left', title='World Population (2017)')

#scale
map.drawmeridians(range(-180,180,40),color='k',dashes=[4, 2],
↪  labels=[0,0,1,0],fontsize=12,zorder=1)
map.drawparallels(range(-90,90,20),color='k',dashes=[4, 2],
↪  labels=[1,0,0,0],fontsize=12,zorder=1)

plt.show()
```

La figure 7.3 montre le résultat obtenu.

FIGURE 7.3 – Population mondiale (en 2017).



Commentaires :

1. On est obligé de passer par les `shapes` et les `collections` de `matplotlib`[2].
2. L'écriture est volontairement répétitive pour permettre aux débutants de comprendre et de maîtriser la chaîne qui mène à la figure 7.3 .
3. On peut évidemment améliorer le script. Dès que vous voyez une série de commandes qui se répètent en changeant juste un numéro vous pouvez améliorer l'écriture à l'image de la ligne 66 du script qui ramasse la création d'une liste de `Patch()` en une ligne. Ceci étant dit l'objectif est de faire des cartes fonctionnelles et propres avec des outils libres pas de faire le plus beau script de la terre.
4. Dès que vous aurez pris un peu d'assurance vous vous rendrez compte que pouvoir tracer des cartes en ayant accès à toutes les fonctionnalités de `matplotlib` est tout simplement fantastique (je sais je me répète).

7.5 Les shapefiles comportant une projection

le problème se complique un peu si le shapefile dont vous disposez est déjà projeté. Afin de pouvoir l'utiliser dans `basemap` il faut le reconvertir on lon/lat ce qui se fait en reprojetant dans la projection Plate carrée.

Le plus simple (à ma connaissance) consiste à

1. charger votre fichier dans `geopandas` une extension de pandas aux données géospatiales ;
2. le convertir dans votre nouvelle projection ;
3. sauvegarder votre nouveau fichier.

```
1  #!/usr/bin/env python2
2  # -*- coding: utf-8 -*-
3  """
4  bm_9.py
5
6  Convert shapefile using Geopandas
7  """
8  #libraries
9  import geopandas
10
11 #read the file
12 deps=geopandas.read_file("/home/metivier/latex/cours/sig/scripts/DEPARTEMENTS/DEPARTEMENT.shp")
13
14
15 print deps.head()
16 print deps.crs
17 print "================"
18
19 #all is here the Plate carree definition
20 d2=deps.to_crs({'init': 'epsg:4326'})
21 print d2.head()
22 print d2.crs
23 print "=================="
24
25 #test
26 print deps.geometry[deps['CODE_DEPT']=='01']
27 print d2.geometry[d2['CODE_DEPT']=='01']
28
29 #save the file
30 d2.to_file("/home/metivier/latex/cours/sig/scripts/DEPARTEMENTS/departement_latlon.shp")
```

Vous noterez que la définition de la projection plate carrée à la ligne 20 est nouvelle. On passe un dictionnaire avec une clé `init` propre à la fonction et sa valeur `epsg:4326`. l'EPSG (pour European Petroleum Survey Group[3]) est une bibliothèque qui référence des centaines de systèmes de coordonnées et de projection. La projection EPSG 4326 correspond à la pate carrée. Le fichier ainsi obtenu peut alors être traité en suivant ce que nous venons de faire à la section précédente.

2. https://matplotlib.org/examples/shapes_and_collections/artist_reference.html
3. www.epsg.org

7.6 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 géometrie correspondante,
4. créer l'objet et le remplir de son attribut et de sa géométrie.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_10.py

Write data to a shapefie
"""
#import libraries
from osgeo import ogr,osr
import sys, os

# set the spatial reference
spatialReference=osr.SpatialReference()
spatialReference.SetWellKnownGeogCS('WGS84')

# create the shapefile
driver = ogr.GetDriverByName('ESRI Shapefile')
dstFile = driver.CreateDataSource('IPGinfo.shp')
if dstFile is None:
        print "could not create file"
        sys.exit(1)

# create the layer with the spatial reference defined above
layer = dstFile.CreateLayer("IPGInfo",spatialReference)

# create the field where we will store the name of places
fieldDef = ogr.FieldDefn("Name",ogr.OFTString)
fieldDef.SetWidth(50)
layer.CreateField(fieldDef) # add the field definition to the layer

"""
 create the two features
"""
# the IPGP feature
featureDefn=layer.GetLayerDefn()
feature = ogr.Feature(featureDefn)
feature.SetField('Name','IPGP')

#create the feature's geometry
point=ogr.Geometry(ogr.wkbPoint)
point.AddPoint(48.844855,2.356685)
feature.SetGeometry(point)

#add the feature to the layer
layer.CreateFeature(feature)
# destroy a feature when finished so it is saved
feature.Destroy()

#do it again for UFR
featureDefn=layer.GetLayerDefn()
```

```
feature = ogr.Feature(featureDefn)
feature.SetField('Name','UFR')
point=ogr.Geometry(ogr.wkbPoint)
point.AddPoint(48.827646,2.380656)
feature.SetGeometry(point)
layer.CreateFeature(feature)
feature.Destroy()
#same for the file a bit like close
dstFile.Destroy()
```

Une fois que c'est fait on peut vérifier que tout va bien en utilisant le script du §7.3

8
Les rasters

8.1 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 example 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.

8.1.1 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.

Voici un exemple avec un fichier jpeg représentant une image Landsat prétraitée.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_11.py

Get information fom a georeferenced image
"""

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from osgeo import gdal
from osgeo import osr
import pyproj
from numpy import linspace
from numpy import meshgrid

ds = gdal.Open("./raster/Aquitaine/LC82000292015341LGN00.jpg")
data = ds.ReadAsArray()


# GET PROJECTION INFORMATION
proj=ds.GetProjection()
inproj = osr.SpatialReference()
inproj.ImportFromWkt(proj)
print "Projection"
print inproj.ExportToProj4()


```

```
29
30 # GET MAP LIMITS
31 geotransform = ds.GetGeoTransform()
32 #upper left corner
33 originX = geotransform[0]
34 originY = geotransform[3]
35 print "upper left corner"
36 print originX, originY
37 #pixel size
38 pixelWidth = geotransform[1]
39 pixelHeight = geotransform[5]
40 print "Pixel size"
41 print pixelWidth, pixelHeight
42
43 #width and height of the image
44 w=abs(pixelWidth * ds.RasterXSize)
45 h=abs(pixelHeight * ds.RasterYSize)
46 print "Width and height"
47 print w,h
48
49 #coordinate of the image center
50 UTM_X=originX+w/2
51 UTM_Y=originY-h/2
52
53
54 #CONVERT THE COORDINATES OF THE CENTER TO LON/LAT
55 srcProj=pyproj.Proj(inproj.ExportToProj4()) # image projection
56 dstProj=pyproj.Proj(proj="longlat",ellps="WGS84",datum="WGS84") # Plate Carree
57 Long,Lat=pyproj.transform(srcProj,dstProj,UTM_X,UTM_Y)
58 print "Original Center coordinates"
59 print UTM_X, UTM_Y
60 print "Converted Center coordinates"
61 print Long, Lat
```

Qui nous donne ceci en sortie

```
Projection
+proj=utm +zone=30 +datum=WGS84 +units=m +no_defs
upper left corner
558270.0 5058330.0
Pixel size
30.0 -30.0
Width and height
230130.0 234330.0
Original Center coordinates
673335.0 4941165.0
Converted Center coordinates
-0.815784938395 44.6029476328
```

Commentaires : Le traitement d'images raster fait appel à une librairie nommée `GDAL (Geospatial Data Abstraction Layer)` une des librairies les plus importantes de traitement de données géospatiales, qui se trouve au coeur de n'importe quel logiciel de cartographie. Il existe une version python (un wrapper) qui nous simplifie considérablement la tâche ! On peut alors se servir des informations pour représenter notre belle image avec en prime la belle ville de Bordeaux (figure 8.1).

```
1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 """
4 bm_12.py
5
6 Plot a georeferenced jpeg image
7 Add a point location and its name
```

FIGURE 8.1 – Landsat TM8 des landes d'Aquitaine.

```
8   """
9
10  from mpl_toolkits.basemap import Basemap
11  import matplotlib.pyplot as plt
12  from osgeo import gdal
13  from osgeo import osr
14  import pyproj
15  from numpy import linspace
16  from numpy import meshgrid
17
18  ds = gdal.Open("./raster/Aquitaine/LC82000292015341LGN00.jpg")
19  data = ds.ReadAsArray()
20
21  # map limits
22  geotransform = ds.GetGeoTransform()
23  originX = geotransform[0]
24  originY = geotransform[3]
25  print originX, originY
26  pixelWidth = geotransform[1]
27  pixelHeight = geotransform[5]
28  print pixelWidth, pixelHeight
29
30
31  w=abs(pixelWidth * ds.RasterXSize)
32  h=abs(pixelHeight * ds.RasterYSize)
33
34  UTM_X=originX+w/2
35  UTM_Y=originY-h/2
36
37  srcProj=pyproj.Proj(proj="utm",zone="30",ellps="WGS84",units="m")
```

```
dstProj=pyproj.Proj(proj="longlat",ellps="WGS84",datum="WGS84")
Long,Lat=pyproj.transform(srcProj,dstProj,UTM_X,UTM_Y)

fig=plt.figure(figsize=(20,20))# because it is a raster the output size is important
map=Basemap(projection='tmerc', lon_0=Long,lat_0=Lat,width=w,height=h)

map.imshow(plt.imread("./raster/Aquitaine/LC82000292015341LGN00.jpg"), origin='upper')

#place Bordeaux to test
x,y=map(-0.568058,44.8409391)
plt.plot(x,y,'ro',ms=12)
plt.text(1.01*x,1.01*y,'Bordeaux',fontsize=12,color='w')

map.drawmeridians(range(-18,18,1),color='r',labels=[0,0,1,1],fontsize=14,zorder=2)
map.drawparallels(range(42,47,1),color='r',labels=[1,1,0,0],fontsize=14,zorder=2)

plt.show()
```

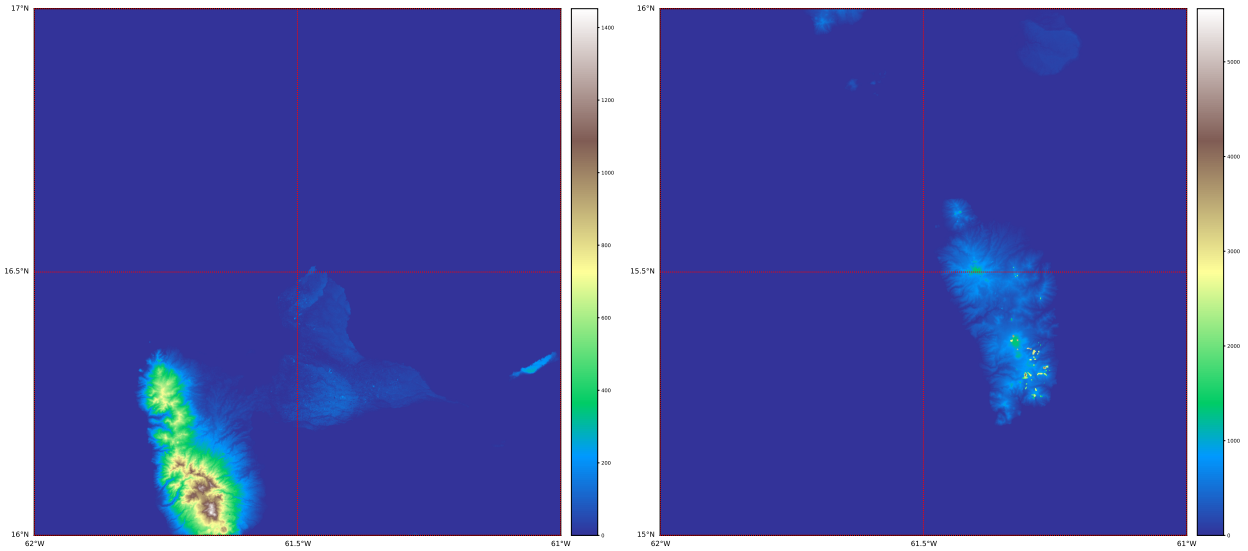
8.1.2 Topographie de la Guadeloupe

- Topographie établie à partir des images du satellite ASTER.
- Résolution de 90m
- Format de fochier Geotiff = ficher 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
- fichiers fournis en tuiles de 1° de côté.
- Pour la Guadeloupe on en a deux...

Procédure :

- On utilise la bilbiothèque GDAL pour ouvrir le fichier
- Extraction des coordonnées des coins
- définition de la Basemap à partir de ces coordonnées
- Représentation graphique.

FIGURE 8.2 – MNT ASTER de la Guadeloupe...en deux bouts




```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_12-1.py

Plot a georeferenced DEM
"""
```

```
8
9  from mpl_toolkits.basemap import Basemap
10 import matplotlib.pyplot as plt
11 from osgeo import gdal
12 from osgeo import osr
13 import pyproj
14 from numpy import linspace
15 from numpy import meshgrid
16
17 import sys
18 sys.path.append('/usr/bin/')
19 import gdal_merge as gm
20
21 ds = gdal.Open("./raster/Guadeloupe/ASTGTM2_N16W062/ASTGTM2_N16W062_dem.tif")
22 data = ds.ReadAsArray()
23
24
25 # map limits
26 geotransform = ds.GetGeoTransform()
27 originX = geotransform[0]
28 originY = geotransform[3]
29 print originX, originY
30 pixelWidth = geotransform[1]
31 pixelHeight = geotransform[5]
32 print pixelWidth, pixelHeight
33
34 w=abs(pixelWidth * ds.RasterXSize)
35 h=abs(pixelHeight * ds.RasterYSize)
36
37 #corners
38 lx=originX
39 ly=originY-h
40 rx=lx+w
41 ry=originY
42
43 print lx,ly
44 print rx,ry
45
46 #center
47 Long=lx+w/2
48 Lat=ly+h/2
49 print Long,Lat
50
51 fig=plt.figure(figsize=(20,20))# because it is a raster the output size is important
52 ax=fig.add_subplot(111)
53 map=Basemap(projection='cyl', llcrnrlon=lx,llcrnrlat=ly,urcrnrlon=rx,urcrnrlat=ry)
54
55
56 c=map.imshow(data,origin='upper',zorder=2,cmap='terrain')
57 cb=map.colorbar(c)
58
59
60 map.drawmeridians([-61,-61.5,-62],color='r',labels=[0,0,0,1],fontsize=14,zorder=4)
61 map.drawparallels([15,15.5,16],color='r',labels=[1,0,0,0],fontsize=14,zorder=4)
62
63 plt.show()
```

Procédure suite :

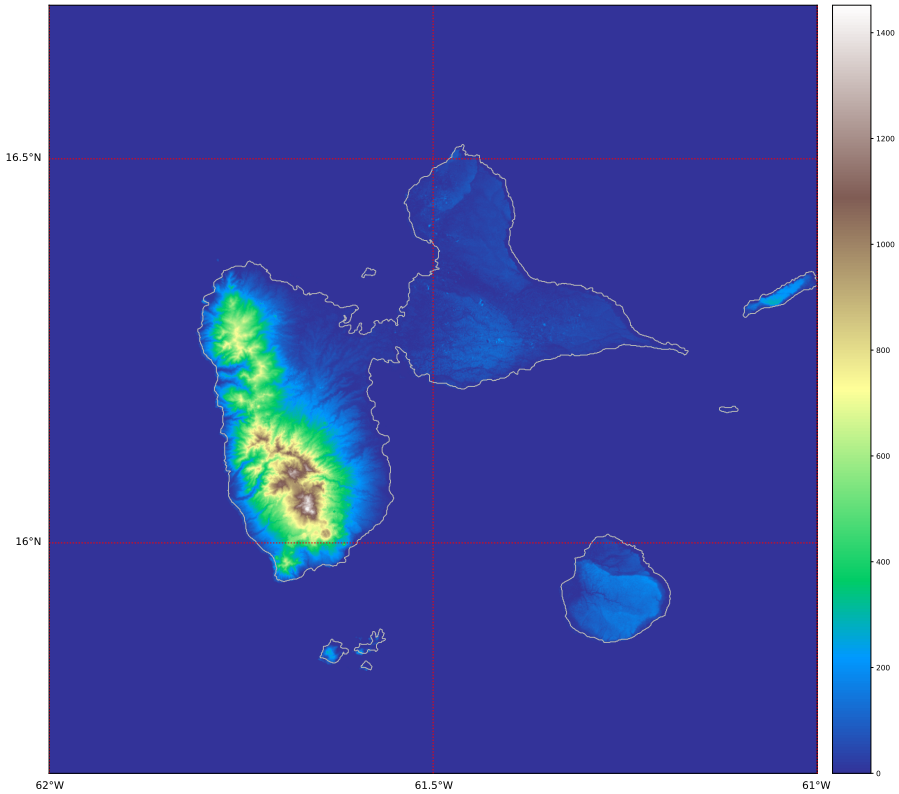
— on colle les deux bouts avec la librairie `gdal_merge` dont la syntaxe est un peu désagréable,

— on coupe l'image finale pour se débarasser de la dominique et des caraïbes au nord de l'île avec la fonction `gdal.Warp`.

Il faut donc rajouter les instructions suivantes au début du script précédent (et adapter les noms de fichier «of course»)

```
import sys
sys.path.append('/usr/bin/')
import gdal_merge as gm

gm.main(['','-n','0','-o','./raster/Guadeloupe/antilles.tif',\
    './raster/Guadeloupe/ASTGTM2_N16W062/ASTGTM2_N16W062_dem.tif',\
    './raster/Guadeloupe/ASTGTM2_N15W062/ASTGTM2_N15W062_dem.tif'])

gdal.Warp("./raster/Guadeloupe/guadeloupe.tif","./raster/Guadeloupeantilles.tif",\
outputBounds=(-62,15.7,-61,16.7))
```

Figure 8.3 – MNT ASTER de la Guadeloupe...en un seul bout !



8.2 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.

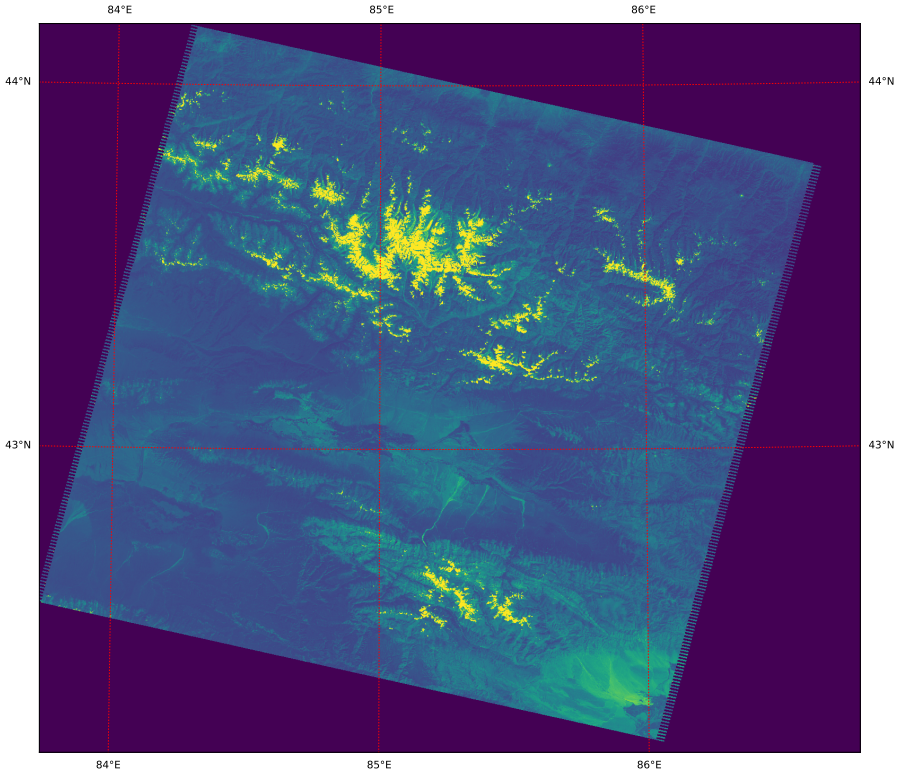
L'application des scripts est assez simple —je vous la laisse en exercice—, attention à la projection lors de la définition de Basemap cependant. On obtient la figure 8.4

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 ;

FIGURE 8.4 – Bassin de Bayanbulak Landsat 5 144030, Canal 10.



2. étirer un peu tout cela ;
3. fondre les images en une seule à trois canaux pour obtenir une image en couleur ;

8.2.1 Histogramme

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_13.py

Open Raster bands from a Geotiff Landsat 5 image
Extract and plot the histograms
"""


import sys
sys.path.append("/home/metivier/src/python/lib")

from osgeo import gdal, osr, ogr
from osgeo.gdalconst import *
import numpy as np
import matplotlib.pylab as pl
import pyproj

# get the GeoTiff driver and register it
driver = gdal.GetDriverByName('GTiff')
driver.Register()


def landsat_hist(fn):

    pl.figure(figsize=(8,11))
    bcount=1
```

```
    for f in fn:
        ds = gdal.Open(f, GA_ReadOnly)
        if ds is None:
            print 'Could not open ' + f
            sys.exit(1)

        #raster size
        cols = ds.RasterXSize
        rows = ds.RasterYSize
        bands = ds.RasterCount

        print bands, rows, cols

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

        print originX, originY, pixelWidth, pixelHeight

        #read band and compute histogram
        band = ds.GetRasterBand(1)
        data = band.ReadAsArray(0, 0, cols, rows)
        h, b = np.histogram(data,255)
        pl.subplot(3,1,bcount)
        pl.semilogy(b[2:250],h[2:250],'r-')
        pl.ylabel('N')
        pl.title(f)
        bcount=bcount+1
    pl.xlabel('pixel value')

    pl.show()



fn=["./raster/China/144030/L5144030_03020060731_B10.TIF",\
"./raster/China/144030/L5144030_03020060731_B20.TIF",\
"./raster/China/144030/L5144030_03020060731_B30.TIF"]
landsat_hist(fn)
```

8.2.2 Etirement 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

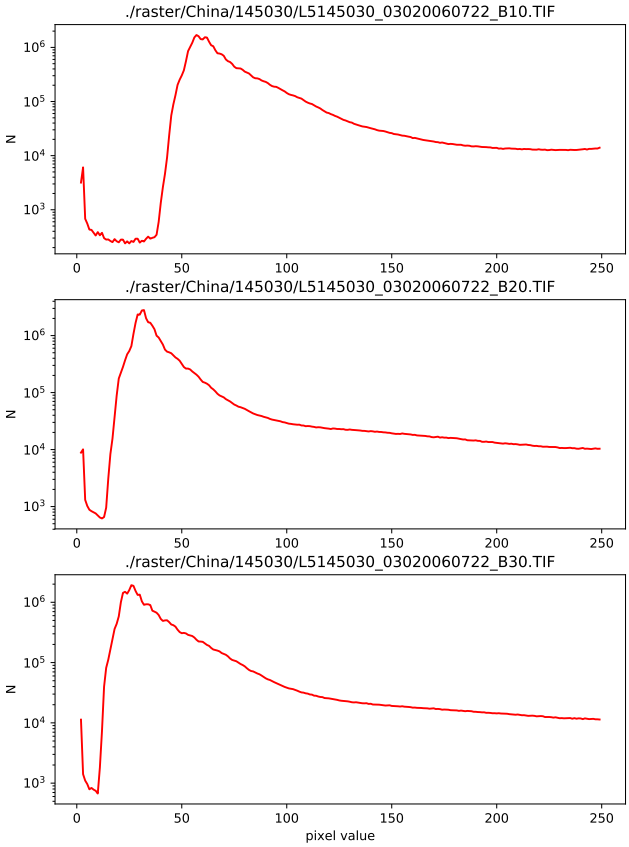
```
data=gdal.Open("./raster/China/144030/L5144030_03020060731_B10.TIF")
gdal.Translate(destName="./raster/China/144030/B10_s.TIF", srcDS=data,scaleParams=[[30, 200,
↪  1, 254]], ,exponents=[0.7])
```

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 claire ; 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` fournit 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

FIGURE 8.5 – Histogramme des trois canaux (10,20,30) de la Landsat 144030




```
import sys
sys.path.append('/usr/bin/')
import gdal_merge as gm

gm.main(['','-o', './raster/China/144030/144030_rgb.tif',\
'-separate','./raster/China/144030/B30_s.TIF',\
'./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 crée antérieurement.

Notre Landsat ressemble maintenant à la figure 8.6. C'est nettement mieux n'est-il-pas ? On pourrait s'en contenter mais non...

8.2.3 Joindre et découper

Le bassin de Bayanbulak correspond en fait aux deux bassins que l'on voit sur la figure 8.6. Pour l'avoir dans son entier il faut fusionner deux images Landsat. La figure 8.7 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 a produire les figures 8.6 et 8.7, la suite prend trois lignes de code...

```
gdal.Warp("./raster/China/144030/144030_rgb_ll.tif",\
    "./raster/China/144030/144030_rgb.tif", dstSRS="+proj=latlong +datum=WGS84",\
```

FIGURE 8.6 – Bassin de Bayanbulak, Landsat 5 144030 ; image composée des trois canaux (R :30,V :20,B :10) étirés

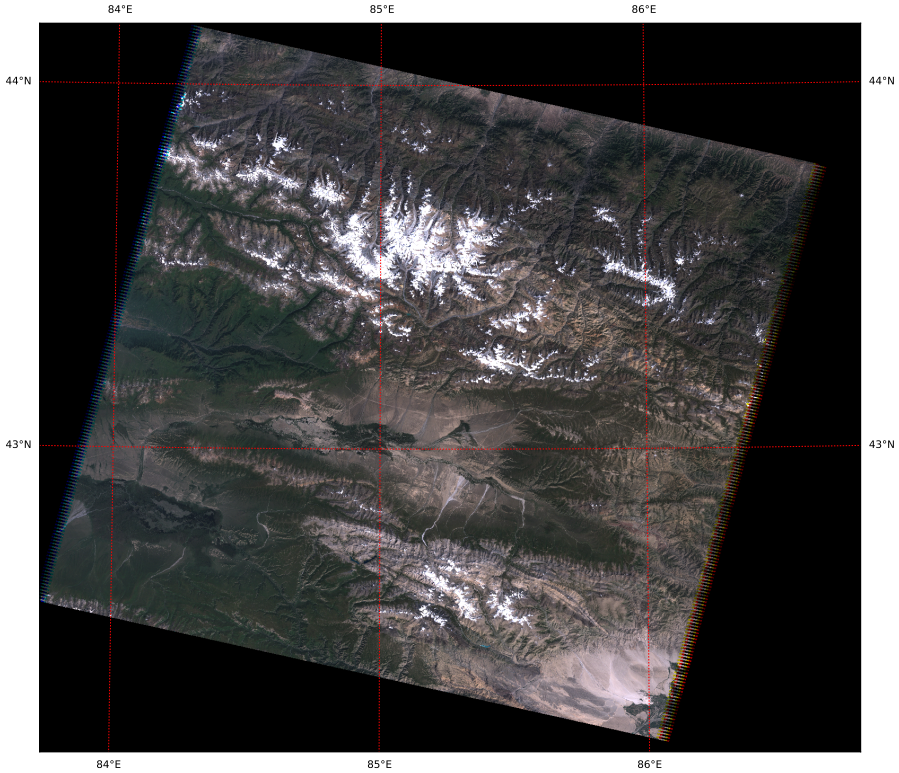
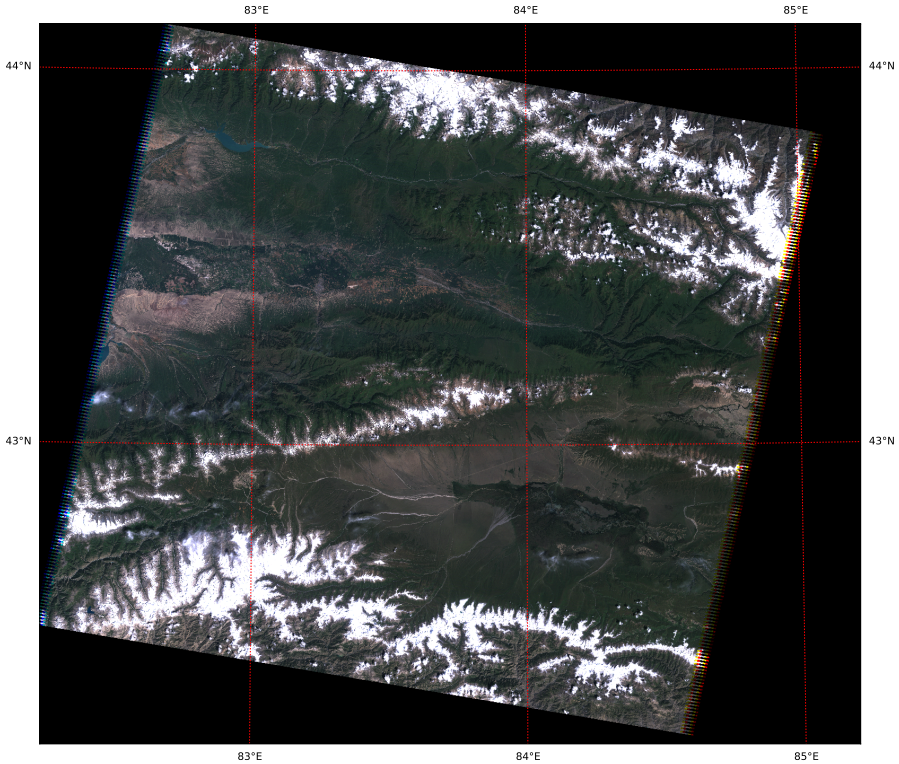


FIGURE 8.7 – Bassin de Bayanbulak Landsat 5 145030, image composée des trois canaux (R :30,V :20,B :10) étirés



```
    outputBounds=(84.25, 42.5, 86.1, 43.5))
gdal.Warp("./raster/China/145030/145030_rgb_ll.tif",\
        "./raster/China/145030/145030_rgb.tif", dstSRS="+proj=latlong +datum=WGS84",\
        outputBounds=(82.5, 42.5, 84.5, 43.5))

gm.main(['','-n','0','-o', './raster/China/bayan.tif',\
 './raster/China/144030/144030_rgb_ll.tif',\
 './raster/China/145030/145030_rgb_ll.tif'])
```

Ne reste plus qu'à créer la carte

```
fig=plt.figure(figsize=(20,10))# because it is a raster the output size is important
map=Basemap(projection='cyl', llcrnrlon=82.5, llcrnrlat=42.5,\
        urcrnrlon=86.1,urcrnrlat=43.5)

map.imshow(plt.imread("./raster/China/bayan.tif"), origin='upper')

map.drawmeridians(range(83,88,1),color='r',labels=[0,0,1,1],fontsize=14)
map.drawparallels([42.5,43,43.5],color='r',labels=[1,1,0,0],fontsize=14)

plt.show()
```

Le script suivant résume l'ensemble des opérations :

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_14.py

(i) Stretch the images according to the histgrams obtained using bm_13
(ii) Merge the bands in one multi-layered raster band
(iii) Reproject and crop the raster
(iv) merge to obtain the final image
(v) plot the map.
"""


import sys
sys.path.append("/home/metivier/src/python/lib")
sys.path.append('/usr/bin/')


from osgeo import gdal, osr, ogr
from osgeo.gdalconst import *
import pyproj
import gdal_merge as gm
import numpy as np
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

# get the GeoTiff driver and register it
driver = gdal.GetDriverByName('GTiff')
driver.Register()

# (i)
# Stretch the histograms
# 144030

data=gdal.Open("./raster/China/144030/L5144030_03020060731_B10.TIF")
gdal.Translate(destName="./raster/China/144030/B10_s.TIF",srcDS=data,scaleParams=[[30, 200,
↪  1, 254]],exponents=[0.7])
data=gdal.Open("./raster/China/144030/L5144030_03020060731_B20.TIF")
```

```
gdal.Translate("./raster/China/144030/B20_s.TIF",data, scaleParams=[[10, 120, 1, 254]],exponents=[0.7])
data=gdal.Open("./raster/China/144030/L5144030_03020060731_B30.TIF")
gdal.Translate("./raster/China/144030/B30_s.TIF",data, scaleParams=[[8, 140, 1, 254]],exponents=[0.7])


# 145030
data=gdal.Open("./raster/China/145030/L5145030_03020060722_B10.TIF")
gdal.Translate(destName="./raster/China/145030/B10_s.TIF",srcDS=data,scaleParams=[[30, 200, 1, 254]],exponents=[0.7])
data=gdal.Open("./raster/China/145030/L5145030_03020060722_B20.TIF")
gdal.Translate("./raster/China/145030/B20_s.TIF",data, scaleParams=[[10, 120, 1, 254]],exponents=[0.7])
data=gdal.Open("./raster/China/145030/L5145030_03020060722_B30.TIF")
gdal.Translate("./raster/China/145030/B30_s.TIF",data, scaleParams=[[8, 140, 1, 254]],exponents=[0.7])

# (ii)
# Merge the bands
gm.main(['','-o', './raster/China/144030/144030_rgb.tif','-separate','./raster/China/144030/B30_s.TIF', './raster/China/144030/B20_s.TIF', './raster/China/144030/B10_s.TIF'])
gm.main(['','-o', './raster/China/145030/145030_rgb.tif','-separate','./raster/China/145030/B30_s.TIF', './raster/China/145030/B20_s.TIF', './raster/China/145030/B10_s.TIF'])


# (iii)
# Crop and reproject the images
gdal.Warp("./raster/China/144030/144030_rgb_ll.tif","./raster/China/144030/144030_rgb.tif", dstSRS="+proj=latlong +datum=WGS84",outputBounds=(84.25, 42.5, 86.1, 43.5))
gdal.Warp("./raster/China/145030/145030_rgb_ll.tif","./raster/China/145030/145030_rgb.tif", dstSRS="+proj=latlong +datum=WGS84",outputBounds=(82.5, 42.5, 84.5, 43.5))

# (iv)
# Merge into the final image
gm.main(['','-n','0','-o', './raster/China/bayan.tif', './raster/China/144030/144030_rgb_ll.tif', './raster/China/145030/145030_rgb_ll.tif'])


# (v)
# Draw the map

ds = gdal.Open("./raster/China/bayan.tif")
data = ds.ReadAsArray()

### map limits
geotransform = ds.GetGeoTransform()
originX = geotransform[0]
originY = geotransform[3]
print originX, originY
pixelWidth = geotransform[1]
pixelHeight = geotransform[5]
print pixelWidth, pixelHeight


w=abs(pixelWidth * ds.RasterXSize)
h=abs(pixelHeight * ds.RasterYSize)
print w,h
UTM_X=originX+w/2
```

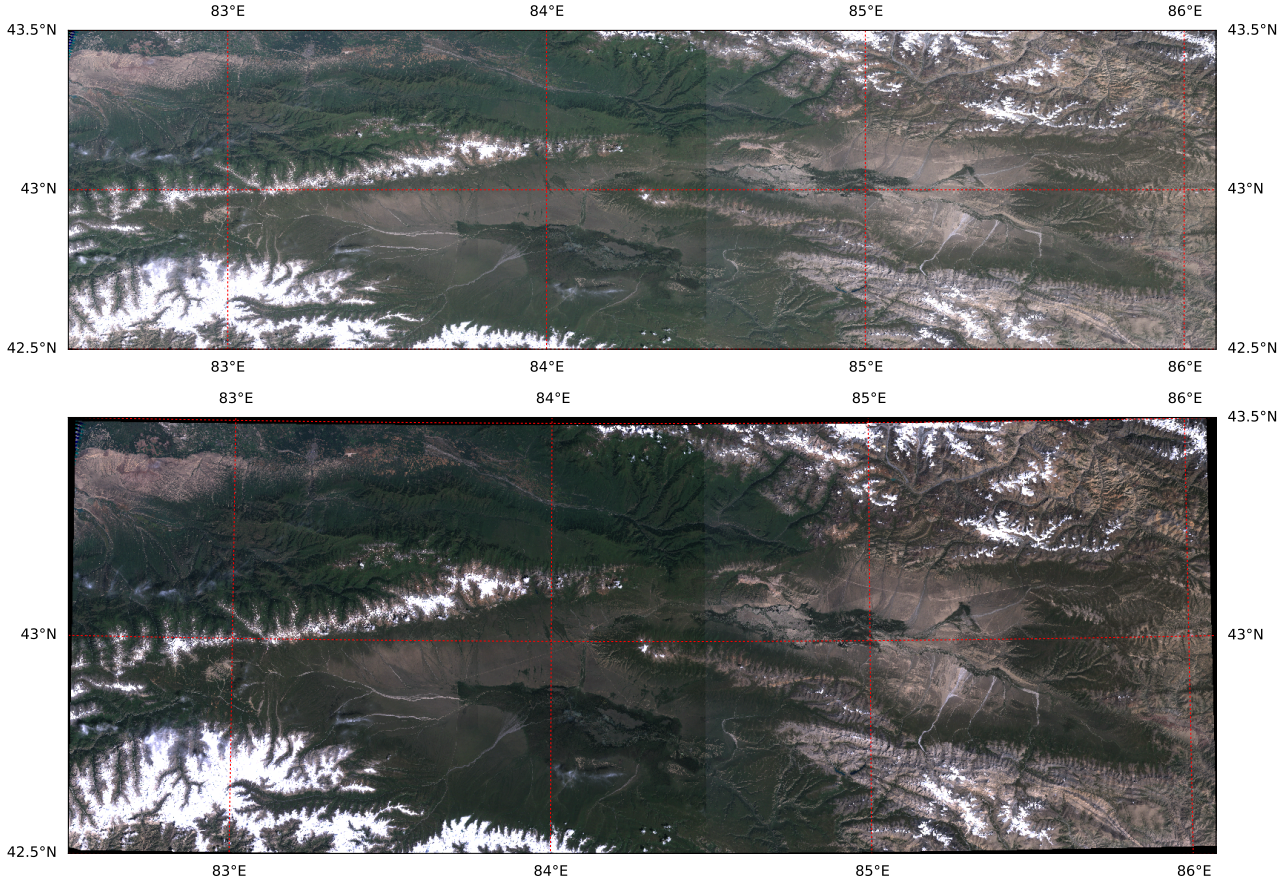
```
UTM_Y=originY-h/2

# get image projection
proj=ds.GetProjection()
inproj = osr.SpatialReference()
inproj.ImportFromWkt(proj)
#conversion of coordinates to long/lat
srcProj=pyproj.Proj(inproj.ExportToProj4())
dstProj=pyproj.Proj(proj="longlat",ellps="WGS84",datum="WGS84")
Long,Lat=pyproj.transform(srcProj,dstProj,UTM_X,UTM_Y)
print Long, Lat

fig=plt.figure(figsize=(20,10))# because it is a raster the output size is important
map=Basemap(projection='cyl', llcrnrlon=82.5, llcrnrlat=42.5,urcrnrlon=86.1,urcrnrlat=43.5)

map.imshow(plt.imread("./raster/China/bayan.tif"), origin='upper')

map.drawmeridians(range(83,88,1),color='r',labels=[0,0,1,1],fontsize=14)
map.drawparallels([42.5,43,43.5],color='r',labels=[1,1,0,0],fontsize=14)

plt.show()
```

Et là, avouons le, le résultat est satisfaisant (figure 8.8). 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 8.8 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`

```
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 8.8 – 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.



9
Exemples d'applications

9.1 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 avions 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.

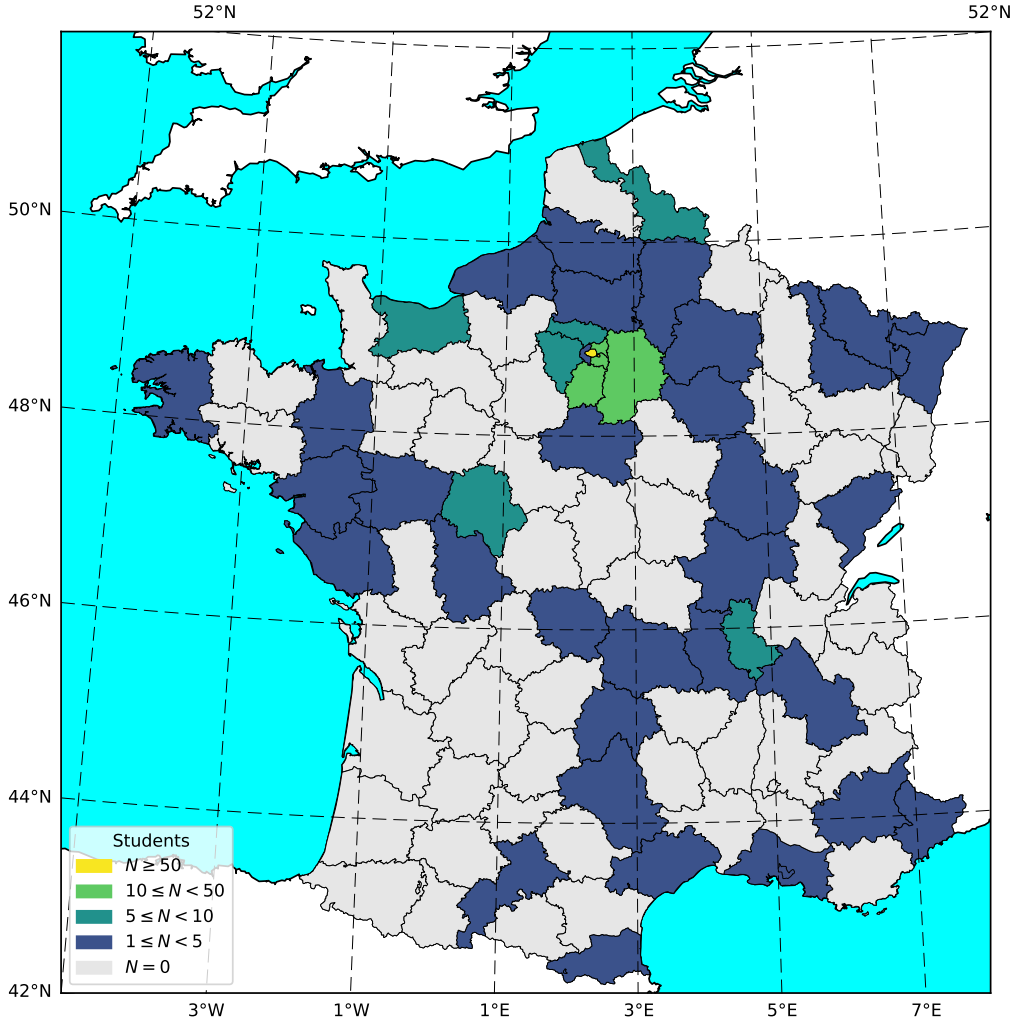
```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_15.py

Map the French départements with the result of a query performed in
the MySQL Parcours database.
"""

#libraries
from mpl_toolkits.basemap import Basemap
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from matplotlib.patches import Patch
from matplotlib.collections import PatchCollection
from matplotlib.patches import PathPatch
import numpy as np
from matplotlib import cm
import MySQLdb # msyql api

#CONNEXION
# name of the database you whish to connect to
base='Parcours'
# establish connection
conn=MySQLdb.connect(host="localhost",user="root",passwd="root",db=base)
# create a curso object to send quaries
```

```
cursor=conn.cursor()
# write the SQL query
query="select * from OneYearBefore "
# execute the query
cursor.execute(query)
# fetch the dataset

#DICTIONNARY AND LIST
# We are going to create a dictionnary with the number of
# students as the value and the departemnt code as the key
# and a list of keys
OYB={}
keys=[]
rows = cursor.fetchall()
for r in rows:
   OYB[r[0]]=float(r[1])
   keys.append(r[0])


#FIGURE
fig=plt.figure()
ax=fig.add_subplot(111)

#MAP
map=Basemap(llcrnrlon=-5,llcrnrlat=42,urcrnrlon=9.,urcrnrlat=52.,\
resolution='i', projection='tmerc', lat_0 = 46, lon_0 = 2.5)

#background maps
map.drawmapboundary(fill_color='aqua')
map.fillcontinents(color='w',lake_color='aqua')
map.drawcoastlines()

#the scale
#the scale
map.drawmeridians(range(-5,10,2),color='k',dashes=[4, 2],\
 labels=[0,0,0,1],linewidth=0.5)
map.drawparallels(range(40,54,2),color='k',dashes=[4, 2],\
labels=[1,0,1,0],linewidth=0.5)

#background department map
map.readshapefile("/home/metivier/latex/cours/sig/scripts/\
DEPARTEMENTS/departement_latlon","deps")

#COLORMAP
#selects and colors the departements
cmap=cm.get_cmap('viridis')
c=(0.9,0.9,0.9)
for info, shape in zip(map.deps_info, map.deps):
    if info['CODE_DEPT'] in keys:
        if OYB[info['CODE_DEPT']]>50:
            c=cmap(0.99)
        elif OYB[info['CODE_DEPT']] >= 10 and OYB[info['CODE_DEPT']] < 50:
            c=cmap(0.75)
        elif OYB[info['CODE_DEPT']] >= 5 and OYB[info['CODE_DEPT']] < 10:
            c=cmap(0.5)
        elif OYB[info['CODE_DEPT']] >= 1  < 5:
            c=cmap(0.25)
    else:
        c=(0.9,0.9,0.9)
    p=Polygon(np.array(shape), closed=True)
    ax.add_collection(PatchCollection([p],edgecolor='k',\
```

```
89      linewidth=0.5, facecolor=c,zorder=2))
90
91  print shape
92
93  #and the legend
94  colors = [cmap(0.99), cmap(0.75), cmap(0.5), cmap(0.25),(0.9,0.9,0.9)]
95  texts = ["$N \geq 50$", "$10 \leq N < 50$",\
96           "$5 \leq N < 10$", "$1\leq N<5$ ","$N=0$"]
97  patches = [ Patch(color=colors[i], label="{:s}".format(texts[i]) )\
98           for i in range(len(texts)) ]
99  plt.legend(handles=patches, loc='lower left',\
100          title='Students')
101
102 #END
103 plt.show()
```

Avis de l'enseignant : le résultat (figure 9.1) mérite largement les efforts consentis durant ce cours !

FIGURE 9.1 – Nombre agrégé d'étudiants par département un an avant leur arrivée à l'UFR STEP



9.2 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'Aquitaine [1]. Notre objectif sera de représenter ces isopièzes dans la région des lacs sur une

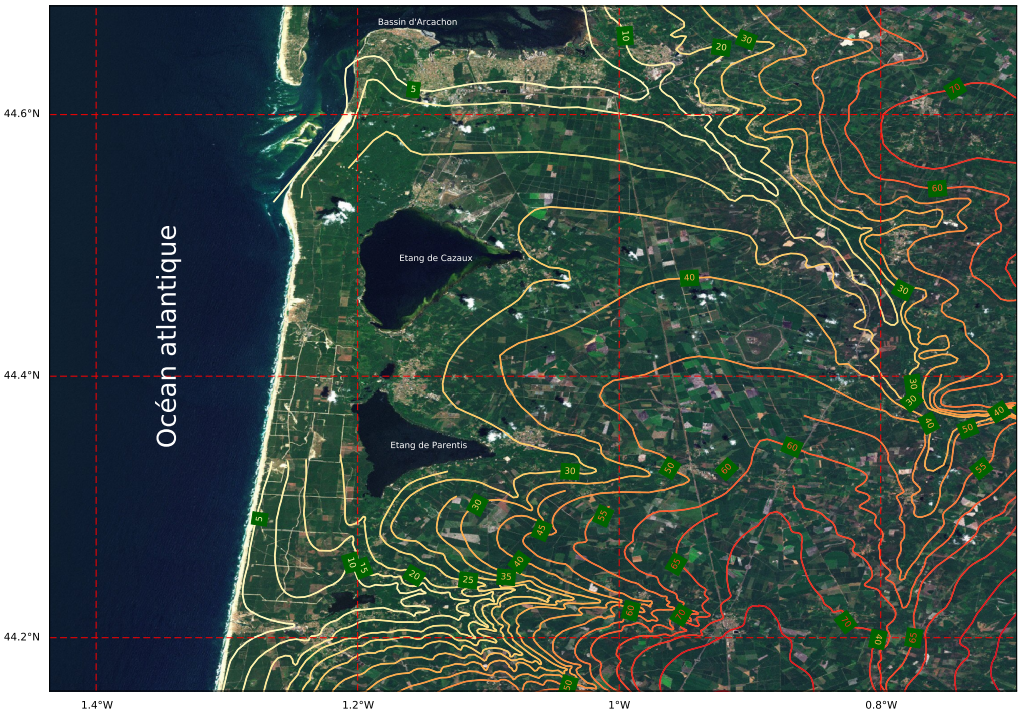
1. Origine des données : SIGES aquitaine http://sigesaqi.brgm.fr/-Les-isopiezes-.html.

image satellite de la région que nous récupérerons de la base d'image du monde mise en ligne gratuitement par ESRI et pour laquelle basemap dispose d'une fonction de récupération `arcgisimage`. La représentation se fera en deux temps : (1) convertir le fichier shp qui n'est pas en lat/lon et (2) projeter le tout sur une belle carte.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_16.py

Draw equipotential (isopieze map) lines of the Plio-quaternary aquifer
on top of a satellite view of the Lakes of Landes south of Arcachon.
"""

#libraries
import geopandas
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from matplotlib import cm
from liblabels import labelLine, labelLines


def convert_shape():
    #read the file
    piezo=geopandas.read_file("./shapefiles/Aquitaine/Isopiezes_mio-plio-quaternaire.shp")


    print piezo.head()
    print piezo.crs
    print "================"

    #test to get rid of null or empty values in the record (else the conversion fails)
    piezo= piezo[piezo['geometry'].notnull()]
    #all is here the Plate carree definition
    piezo2=piezo.to_crs({'init': 'epsg:4326'})
    print piezo2.head()
    print piezo.crs
    print "=================="


    #save the file
    piezo2.to_file("./shapefiles/Aquitaine/Isopiezes_mio-plio-quaternaire_latlon.shp")


def map_piezo():

    # Define figure size and basemap
    fig=plt.figure(figsize=(20,20))
    ax=fig.add_subplot(111)
    map = Basemap(llcrnrlat=44.159251,llcrnrlon=-1.435541,urcrnrlat=44.683123,
    ↪  urcrnrlon=-0.696010,epsg=4230)

    # Retrive satellite image from ESRI
    map.arcgisimage(service='ESRI_Imagery_World_2D', xpixels = 1500, verbose= True)

    # Open shapefile
    map.readshapefile("./shapefiles/Aquitaine/Isopiezes_mio-plio-quaternaire_latlon",
    ↪  "piezo",color='w')

    # retrieve the polylines and plot them using different colors
    cmap=cm.get_cmap('YlOrRd')
    for info, shape in zip(map.piezo_info, map.piezo):
```

```
            c=cmap(float(info['COTE_mNGF'])/100.)
            x,y=zip(*shape)
            ax.plot(x,y,'-',linewidth=2,color=c,label=info['COTE_mNGF'])

    # Label the contours
    labelLines(plt.gca().get_lines(),backgroundcolor='DarkGreen')

    # Print some information
    x,y=map(-1.1681992,44.48807)
    ax.text(x,y,"Etang de Cazaux",color='w')
    x,y=map(-1.1749838,44.345)
    ax.text(x,y,'Etang de Parentis',color='w')
    x,y=map(-1.355,44.5)
    ax.text(x,y,u"Océan atlantique",color='w',rotation=90,fontsize=30)
    x,y=map(-1.184541,44.668606)
    ax.text(x,y,"Bassin d'Arcachon",color='w')


    # Scale
    map.drawmeridians([-1.4,-1.2,-1,-0.8],color='r',dashes=[4, 2],
    ↪  labels=[0,0,0,1],fontsize=12)
    map.drawparallels([44.2,44.4,44.6],color='r',dashes=[4, 2],
    ↪  labels=[1,0,1,0],fontsize=12)


    # Show
    plt.show()
```

FIGURE 9.2 – Hydrogéologie des bassins des étangs de Cazaux et de Parentis. Image satellite : ESRI World Imagery. Hydrogéologie : SIGES Aquitaine.



Commentaires.

1. Pour tracer les `POLYLINE`, la procédure ressemble à celle vue précédement pour les `POLYGON`.
2. Tracer les label sur les lignes demande l'usage d'une bibliothèque spécifique (Annexe C). La bilbiothèque originale est en ligne sur github [2]. Je l'ai adaptée légèrement pour les besoins de cartes avec des lignes qui

2. https://github.com/cphyc/matplotlib-label-lines.

ne suivent pas les formes régulières de fonctions mathématiques. Le principe de ce genre de bibliothèque, comme des scripts précédents d'ailleurs, n'est pas très compliqué. Cela montre assez bien ce qui, dans les faits, se cache derrière les boutons des logiciels comme MapInfo, ArcInfo, QGIS etc.

9.3 Lacs des Landes, seconde manière

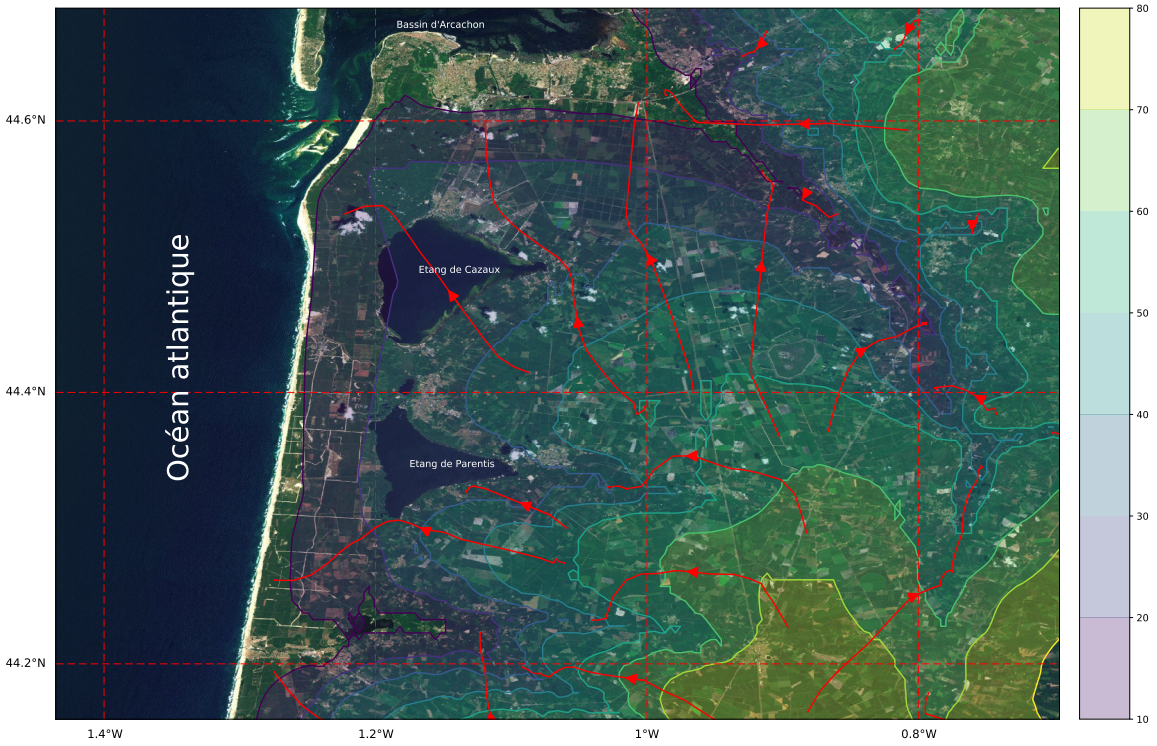
La seconde façon d'utiliser le même jeu de données consiste à interpoler [3] les isopièzes du shapefile sur une grille dans python puis d'utiliser cette grille pour représenter les izopièses en utilisant les fonctions classiques de matplotlib comme `contourf` par exemple. Cette seconde méthode présente 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 9.3). 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.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
bm_17.py

Purpose :
read shapefile
regrid equipotential head lines
calculate streamlines
plot
"""
import geopandas
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.mlab import griddata # this one not the scipy one
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from matplotlib import cm
from liblabels import labelLine, labelLines

# open shapefile
deps=geopandas.read_file("./shapefiles/Aquitaine/Isopiezes_mio-plio-quaternaire_latlon.shp")

print deps.head()

X=[]
Y=[]
Z=[]
for i in range(len(deps["geometry"])):

        try:
                #print deps["geometry"][i].geom_type
                if deps["geometry"][i].geom_type == 'LineString':
                        x,y=deps["geometry"][i].coords.xy
                        z=deps['COTE_mNGF'][i]*np.ones(len(x))
                        X=np.hstack((X,x))
                        Y=np.hstack((Y,y))
                        Z=np.hstack((Z,z))

```

3. 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.

```
                elif deps["geometry"][i].geom_type == 'MultiLineString':
                        for line in deps["geometry"][i]:
                                x,y=line.coords.xy
                                z=deps['COTE_mNGF'][i]*np.ones(len(x))
                                X=np.hstack((X,x))
                                Y=np.hstack((Y,y))
                                Z=np.hstack((Z,z))
        except:
                print i
                pass



xs=np.linspace(min(X),max(X),400)
ys=np.linspace(min(Y),max(Y),400)
xx,yy=np.meshgrid(xs,ys)
zz=griddata(X,Y,Z,xs,ys,interp='linear') # grid data this method does NO EXTRAPOLATION which leads to a masked array

v,u=np.gradient(zz)# calculate gradient BEWARE lines (y-direction) first, columns (x-direction) second


# Define figure size and basemap
fig=plt.figure(figsize=(20,20))
ax=fig.add_subplot(111)
map = Basemap(llcrnrlat=44.159251,llcrnrlon=-1.435541,urcrnrlat=44.683123, urcrnrlon=-0.696010,epsg=4230)

# Retrive satellite image from ESRI
map.arcgisimage(service='ESRI_Imagery_World_2D', xpixels = 1500, verbose= True)

c=ax.contourf(xx,yy,zz,[10,20,30,40,50,60,70,80],alpha=0.3)
ax.contour(xx,yy,zz,[10,20,30,40,50,60,70,80])
ax.streamplot(xx,yy,-u,-v,density=1,color='red',arrowsize=2.5) # streamlines U_Darcy ∝ −grad(h)
map.colorbar(c)}


# Print some information
x,y=map(-1.1681992,44.48807)
ax.text(x,y,"Etang de Cazaux",color='w')
x,y=map(-1.1749838,44.345)
ax.text(x,y,'Etang de Parentis',color='w')
x,y=map(-1.355,44.5)
ax.text(x,y,u"Océan atlantique",color='w',rotation=90,fontsize=30)
x,y=map(-1.184541,44.668606)
ax.text(x,y,"Bassin d'Arcachon",color='w')


# Scale
map.drawmeridians([-1.4,-1.2,-1,-0.8],color='r',dashes=[4, 2], labels=[0,0,0,1],fontsize=12)
map.drawparallels([44.2,44.4,44.6],color='r',dashes=[4, 2], labels=[1,0,1,0],fontsize=12)


# Show
plt.savefig("../figures/shp2grid.pdf",bbox_inches="tight")
plt.show()
```

FIGURE 9.3 – Hydrogéologie des bassins des étangs de Cazaux et de Parentis 2ème version.



Annexes

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 Bio-info 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 qutre opérations : deux créations de variables auxquelles nous affectons une valeur entière et deux calcul que nous enregistrons dans deux autres variables.

```
>>>a=2 # affecte la valeur deux à une variable a
>>>b=3 # affecte la valeur trois à une variable b
>>>c=a+b # affecte la somme de a et b à une variable c
>>>d=a/b # affecte la division de a par b à une variable d
>>>c
5
>>>d
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

```
>>>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/cours-python.pdf

```
>>>x1=2
>>>x2=3
>>>if x1>x2:
...         distance=x1-x2 # ne pas oublier d'indenter
...else:
...         distance=x2-x1
...
>>> distance
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

```
>>>def distance1D(x1,x2): # x1 et x2 sont des arguments
...         if x1>x2:
...                 distance=x1-x2
...         else:
...                 distance=x2-x1
...         return distance
...
```

On peut maintenant appeler notre fonction

```
>>>distance1D(3,8)
>>>5
```

Bon vous venez de galérer pour écrire cinq lignes non ? changeons de technique. lancer gedit et écrire le code dans un fichier que l'on exécutera ensuite

```
# une ligne de commentaire : mon beau programme
# fichier distance1D.py
def distance1D(x1,x2):
        # calcul de la distance sur un axe muni de son repère normé
        if x1>x2:
                distance=x1-x2
        else:
                distance=x2-x1
        return distance
```

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

```
# une ligne de commentaire : mon beau programme
# fichier distance1D.py
def distance1D(x1,x2):
        # calcul de la distance sur un axe muni de son repère normé
        if x1>x2:
                distance=x1-x2
        else:
                distance=x2-x1
        return distance

distance1D(2,7)
```

puis dans une fenêtre vous exécutez

```
> python distance1D.py
```

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

```
# une ligne de commentaire : mon beau programme
# fichier distance1D.py
def distance1D(x1,x2):
        # calcul de la distance sur un axe muni de son repère normé
        if x1>x2:
                distance=x1-x2
        else:
                distance=x2-x1
        return distance

if ___name___='___main___':
        distance1D(2,7)
```

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

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

```
#structure
#import needed libraries
# you have to choose one way OR the other not both

# first way
from matplotlib.pylab import *

# second way
import matplotlib.pylab as py

# in the first case you call a function from the pylab library by
x=arange(10)
#in the second case by
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» à MATLAB. Si vous tapez

```
from pylab import *
```

ceci chargera automatiquement toutes les fonctions de `pyplot` et de `numpy`

A.2.2 functions

```
#define the function
def square(x):
        return x*x

#then use it Note the indentation !
y=square(2)
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).

```
class person:
        def __init__(self,name="bobo",age="0"):
                self.name=name
                self.age=age
                self.talk()

        def speak(self):
                print "my name is %s" % self.name
                print "I am %s years old" % self.age


# use the class
levieux=person("Francois",47)
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.

`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 : $y = x\cos(x)$

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).

```
# -*- coding: utf-8 -*-
"""
pm_A1.py

plox xcos(x)
"""
#import needed libraries
from matplotlib.pylab import *

#define the data
x=linspace(0,10,100)
y=x*cos(x)

#plot the result
figure() #define the figure
plot(x,y,'r-') #make the plot
xlabel('x',fontsize=14) #add labels
ylabel('y',fontsize=14)
legend(["$y=x \cos(x)$"],loc="upper left") # add legend including LaTeX commands
savefig("../figures/pm_A1.pdf",bbox_inches='tight') # save figure to file
show() # show the figure
```

A.3.2 Surface libre d'une nappe

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

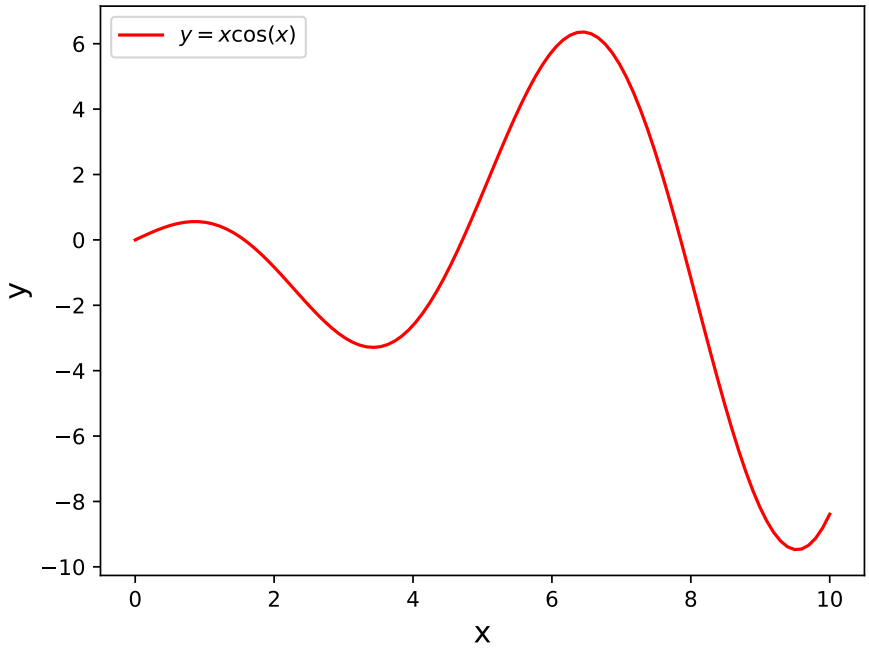


FIGURE A.1 – $x\cos(x)$

Le problème est alors défini par

$$\nabla^2 h^2 = 0 \tag{A.1}$$
$$-Kh\partial_x h(0,t) = q \tag{A.2}$$
$$h(L) = 0 \tag{A.3}$$

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

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

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

```
# -*- coding: utf-8 -*-
"""
pm_A2.py

Solution of a steady 1D flow using Dupuit approximation
for different values of hydraulic conductivity
Boundary conditions are flux a qimposed at 0 and zero depth y at 1.
"""
# Import libraries
from matplotlib.pylab import *

# Define the problem
L=0.8 # "true" length
x=linspace(0,L,100) # x value array
q=1./6000 # discharge m2/s
K=[0.1,0.075,0.05,0.025,0.01] # hydraulic conductivity

# Prepare the figure
figure()
lgd=[] #legend

for k in K:
```

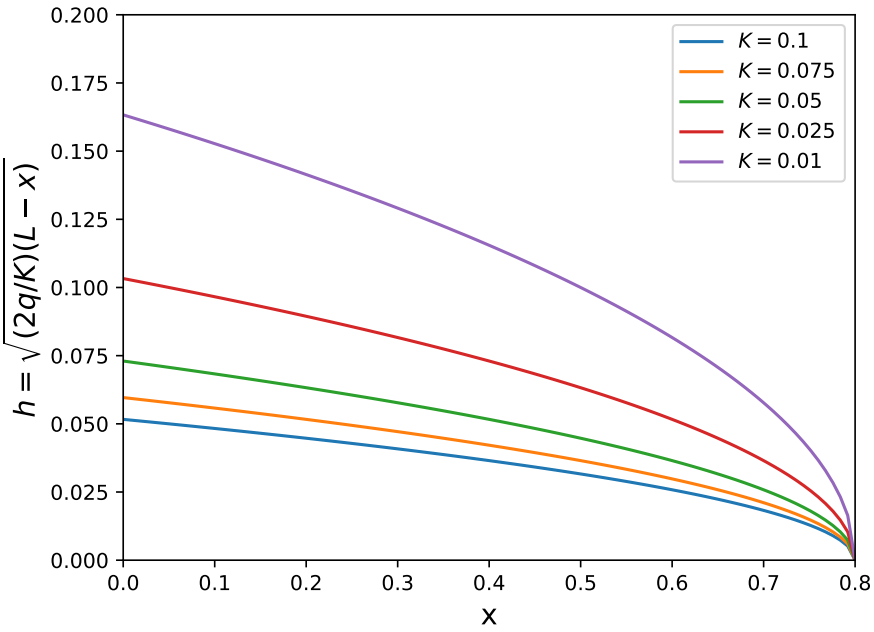


FIGURE A.2 – Profils d'une nappe en état stationnaire pour différentes conductivités hydrauliques.

```
    y=sqrt((2*q/k)*(L-x)) # analytic solution for the water thickness
    st="$ K=%s $" % (k) # add the conductivity to the legend
    lgd.append(st)
    plot(x,y)     #plot the result

# Figure labels, legend and axis
xlabel('x',fontsize=14)
ylabel('$h=\sqrt{(2q/K)(L-x)}$',fontsize=14)
legend(lgd,loc="upper right") # including LaTeX labels
axis([0,0.8,0,0.2])

# save and display
savefig("../figures/pm_A2.pdf",bbox_inches='tight')
show()
```

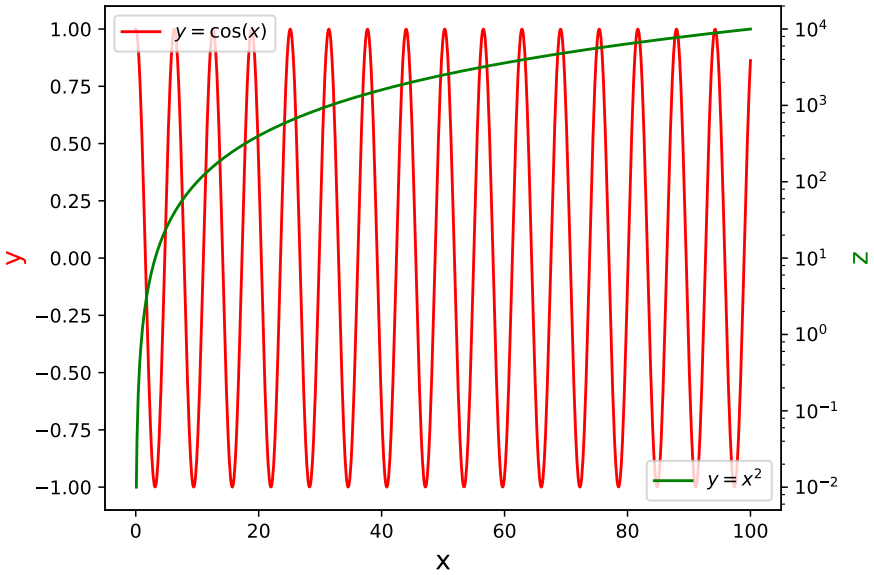
A.4 L'utilisation des axes

le script suivant contient deux fonctions qui font appel à une libraire 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.

```
# -*- coding: utf-8 -*-
"""
pm_A3.py

Using axes
"""

#import needed libraries
from matplotlib.pylab import *

def xcosx():
    #define the data
```

```
    x=linspace(0,10,100)
    y=x*cos(x)
    y2=x*sin(x)
    #plot the result
    fig=figure() #define the figure
    ax=fig.add_subplot(211) #create the axes and add a subplot
    ax2=fig.add_subplot(212)
    ax.plot(x,y,'r-') #make the plot
    ax2.plot(x,y2,'g-')
    ax.set_xlabel('x',fontsize=14) #add labels
    ax.set_ylabel('y',fontsize=14)
    ax.legend(["$y=x \cos(x)$"],loc="upper left") # add legend including LaTeX commands
    ax2.legend(["$y=x \sin(x)$"],loc="upper left") # add legend including LaTeX commands



def two_axes():
    #define the data
    x=linspace(0,100,1000)
    y=cos(x)
    z=x**2
    #plot the results
    fig2=figure() #define the figure
    ax21=fig2.add_subplot(111) #create the axes and add a subplot
    ax22=ax21.twinx() # create a new y-axis but keep the same x-axis
    ax21.plot(x,y,'r-',label='$y=\cos(x)$') #make the plot
    ax22.semilogy(x,z,'g-', label="$y=x^2$")
    ax21.set_xlabel('x',fontsize=14) #add labels
    ax21.set_ylabel('y',color='r',fontsize=14)
    ax22.set_ylabel('z',color='g',fontsize=14)
    ax21.legend(loc="upper left") # add legend including LaTeX commands
    ax22.legend(loc="lower right")
    savefig('../figures/pm_A3.pdf', bbox_inches='tight')
    show() # show the figure
```

FIGURE A.3 – Doubler un axe avec axes.

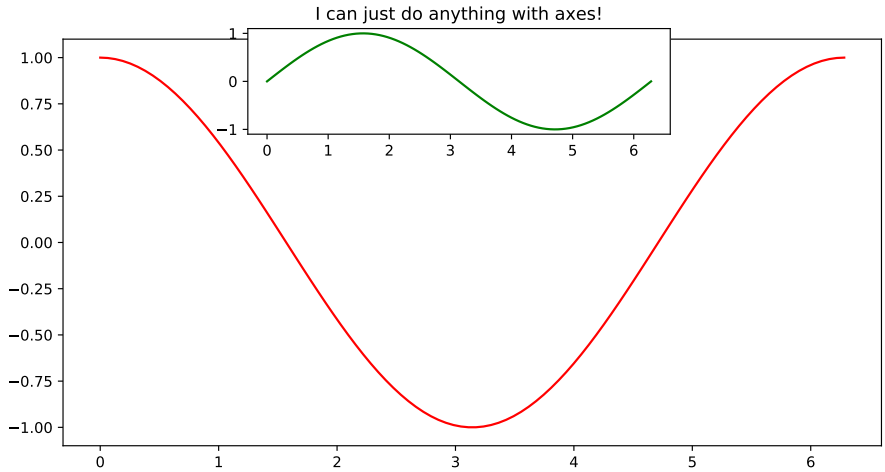


Une figure peut-être considérée comme une page blanche. Les axes permettent de positionner un repère sur la page de la figure à l'endroit désiré comme le montre le script suivant

```
import matplotlib.pyplot as plt
import numpy as np
```

```
fig=plt.figure(figsize=(10,5))
ax=fig.add_subplot(111)

x=np.linspace(0,2*np.pi,100)
y=np.cos(x)

ax.plot(x,y,'r-')

ax2=fig.add_axes([0.3,0.7,0.4,0.2])
ax2.plot(x,np.sin(x),'g-')
ax2.set_title("I can just do anything with axes!")

plt.savefig("../figures/pm_A3_1.pdf",bbox_inches='tight')
plt.show()
```

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



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 souhaiter prendre le contrôle de votre figure et l'adapter réellement à vos besoins.

Enfin quand vous utilisez des subplot() comme

```
fig=figure()
ax1=fig.add_subplot(211)
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.

A.5 Collections et formes géométriques

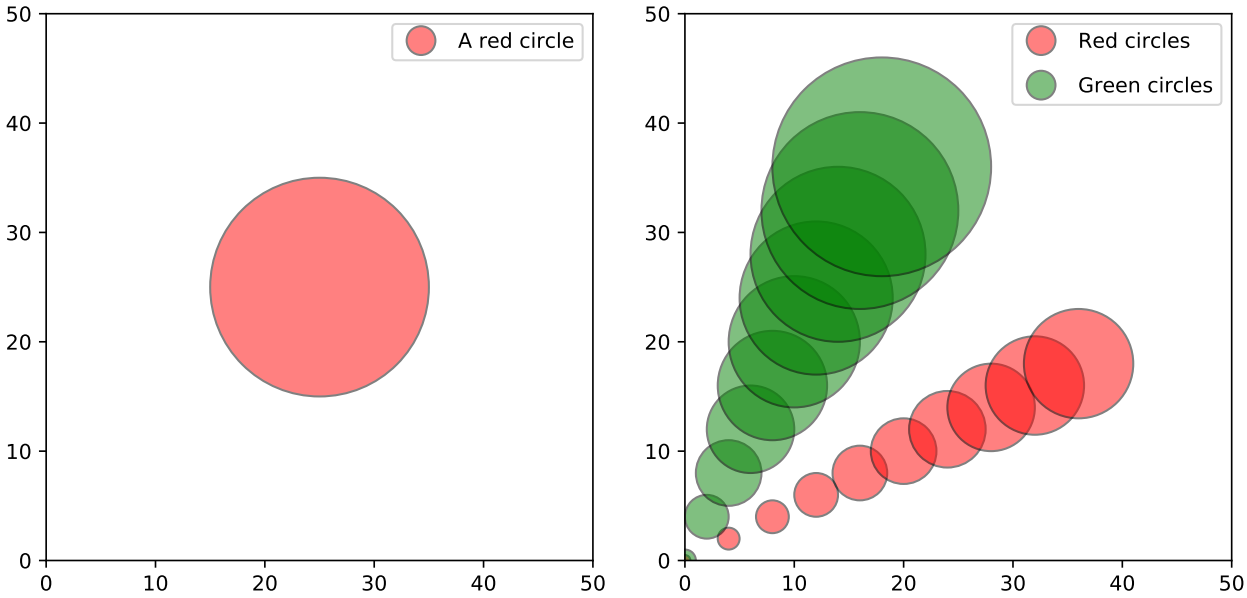
Un exemple parmi d'autres mais important pour nous par la suite : l'utilisation de formes. Nous allons dessiner des cercles (figure A.5)...

```
# -*- coding: utf-8 -*-
"""
pm_A4.py

Drawing Circle
"""


import matplotlib.pyplot as plt
import matplotlib.patches as pch
```

```
import matplotlib.collections as col
from matplotlib.lines import Line2D

def first():
    fig=plt.figure()
    ax=fig.add_subplot(111)

    #create the circle
    c=plt.Circle([25,25],radius=10,fc='r',ec='k',alpha=0.5)
    #add it to the figure
    ax.add_artist(c)

    #set axis form
    ax.axis('square')
    #Fix the limits. Axes will not know the limits from the Circle instance
    ax.set_ylim(0,50)
    ax.set_xlim(0,50)

    #Tricky Legend. Add a handle because legend can not get labels from a Circle (or a
    ↪  Rectangle or a Polygon)
    hd=[Line2D([0],[0], marker='o', markersize=14, markerfacecolor='r', markeredgecolor='k',
    ↪  lw=0, alpha=0.5)]
    ax.legend(hd,['A red circle'])

    #save and show
    plt.savefig("../figures/Circle.pdf",bbox_inches='tight')
    plt.show()

def second():
    fig=plt.figure()
    ax=fig.add_subplot(111)

    #create the red circles
    circles=[]
    for i in range(10):
        c=pch.Circle([4*i,2*i],radius=0.5*(i+1))
        circles.append(c) # add the circle to a list of circles
    #create a collection of circles
    ccol=col.PatchCollection(circles, edgecolor='k',facecolor='r',alpha=0.5,label='red')
    #add them to the figure
    ax.add_collection(ccol)

    # same for the green circles
    circles=[]
    for i in range(10):
        c=pch.Circle([2*i,4*i],radius=(i+1))
        circles.append(c)
    ccol=col.PatchCollection(circles, edgecolor='k',facecolor='g',alpha=0.5,label='green')
    ax.add_collection(ccol)

    #set the axis.
    ax.axis('square')
    ax.set_ylim(0,50)
    ax.set_xlim(0,50)

    #Tricky legend
    #create the handles using a Line2D object
    hd=[Line2D([0],[0], marker='o', markersize=14, markerfacecolor='r', markeredgecolor='k',
    ↪  lw=0, alpha=0.5), Line2D([0],[0], marker='o', markersize=14, markerfacecolor='g',
    ↪  markeredgecolor='k', lw=0,alpha=0.5)]
    #create the legend calling the handles and adding labels.
```

```
    ax.legend(hd,['Red circles','Green circles'],labelspacing=1.2)

    #save and show
    plt.savefig('../figures/Circles.pdf',bbox_inches='tight')
    plt.show()
```

FIGURE A.5 – Utilisation des objets `collections` et `patches`



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 servirons 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`.

```
# -*- coding: utf-8 -*-
"""
pm_A5.py

Different ways to build a legend
"""
#import needed libraries
from matplotlib.pylab import *
from matplotlib.lines import Line2D

def leg_1():
        """Adding labels"""
        #define the data
        x=linspace(0,10,100)
        y=x*cos(x)
        z=x*sin(x)

        figure() #define the figure
        plot(x,y,'r-',label='$x\cos(x)$') #LABELS PLACED INSIDE THE PLOT COMMANDS
        plot(x,z,'g-',label='$x\sin(x)$')

```

```
        xlabel('x',fontsize=14) #add labels
        ylabel('y',fontsize=14)

        #LEGEND CALL
        legend(loc="upper left") # add legend including LaTeX commands

        show() # show the figure

def leg_2():
        """Adding labels second version"""
        #define the data
        x=linspace(0,10,100)
        y=x*cos(x)
        z=x*sin(x)

        figure() #define the figure
        plot(x,y,'r-')
        plot(x,z,'g-')

        xlabel('x',fontsize=14) #add labels
        ylabel('y',fontsize=14)

        #LEGEND CALL
        legend(labels=['$x\cos(x)$','$x\sin(x)$'],loc="lower left") # add legend including
        ↪  LaTeX commands

        show() # show the figure

def leg_3():
        """Retrieving axes, handles and labels"""
        #define the data
        x=linspace(0,10,100)
        y=x*cos(x)
        z=x*sin(x)

        fig=figure() #define the figure
        ax=fig.add_subplot(111)
        ax.plot(x,y,'r-',label='$x\cos(x)$')
        ax.plot(x,z,'g-',label='$x\sin(x)$')

        xlabel('x',fontsize=14) #add labels
        ylabel('y',fontsize=14)

        #LEGEND CALL
        handles,labels=ax.get_legend_handles_labels()
        legend(handles,labels,bbox_to_anchor=(0.5,1.1),ncol=2,loc='center') # add legend
        ↪  including LaTeX commands

        show() # show the figure

def leg_4():
        """"Phantom legend: very useful to represent collections and group labels"""

        fig=figure() #define the figure
        ax=fig.add_subplot(111)

        # Create a phantom plot
        for i in range(4):
                plt.scatter([], [], s=np.exp(i+1),
                ↪  edgecolor='k',facecolor='none',label=str(i+1))

```

```
80
81          # Get the legend handles andlabels
82          h, l = plt.gca().get_legend_handles_labels()
83          # Add a new handle and label
84          h.append(Line2D([0],[0],color='r', lw=3))
85          l.append('a red line !')
86
87          ax.legend(h, l,title="Phantoms",labelspacing=2,borderpad=2,frameon=True,
            ↪  ncol=5,loc=2)
88
89          show()
90
```

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 draîne 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 autre 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 tables `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 enregistement 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 en 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 à 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 ?

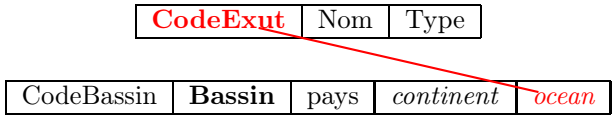


FIGURE B.1 – Lien entre deux tables

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 pouquoi ?

B.1.1 Requètes simples sur une table

1. Combiens d'enregistrements dans la table TLGemsInfoBassins ?
2. Combiens d'enregistrements dans la table TblFlux ?
3. Combiens 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 définit ?

B.1.2 Requètes 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 ordronnée par Exutoire ?
5. Combien de bassins par type d'exutoire ?
6. Combien de bassins en Amérique du sud ?
7. Quel pourcentage de la surperficie 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 francaises
12. Quel est le rapport de ces flux pour chaque continent ? Ordonnez le résultat pra 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 requètes emboitées

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 ?

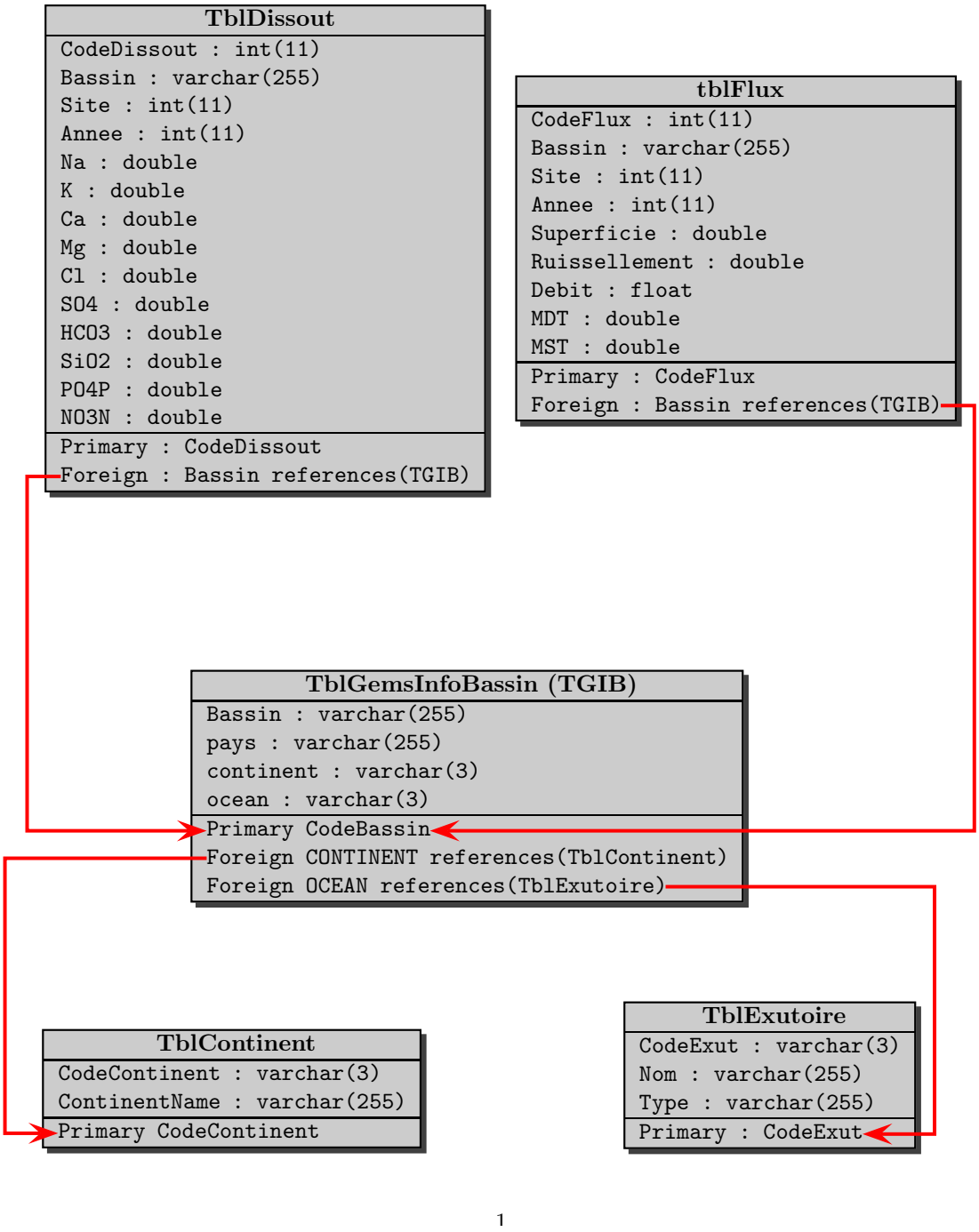


FIGURE B.2 – Schéma de la Base GEMS

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 2017.

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 emboitées

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 ?

C
LabelLines

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 26 10:56:16 2018

modified from the matplotlib-label-lines
https://pypi.org/project/matplotlib-label-lines/

to plot inline labels from labeled polyline shapefiles to plot in basemap
added and change in x value if the original one is outside xaxis range
added a test for y at the end before plotting the data
works ok.
"""
from math import atan2,degrees
import numpy as np


#Label line with line2D label data
def labelLine(line,x,label=None,align=True,**kwargs):

    ax = line.axes
    xdata = line.get_xdata()
    ydata = line.get_ydata()

    if (x < xdata[0]) or (x > xdata[-1]):
        print('x label location is outside data range! changing x')
        x=np.median(xdata)

    #Find corresponding y co-ordinate and angle of the line
    ip = 1
    for i in range(len(xdata)):
        if x < xdata[i]:
            ip = i
            break

    y = ydata[ip-1] + (ydata[ip]-ydata[ip-1])*(x-xdata[ip-1])/(xdata[ip]-xdata[ip-1])

    if not label:
        label = line.get_label()

    if align:
        #Compute the slope
        dx = xdata[ip] - xdata[ip-1]
        dy = ydata[ip] - ydata[ip-1]
        ang = degrees(atan2(dy,dx))

        #Transform to screen co-ordinates
```

```
        pt = np.array([x,y]).reshape((1,2))
        trans_angle = ax.transData.transform_angles(np.array((ang,)),pt)[0]

    else:
        trans_angle = 0

    #Set a bunch of keyword arguments
    if 'color' not in kwargs:
        kwargs['color'] = line.get_color()

    if ('horizontalalignment' not in kwargs) and ('ha' not in kwargs):
        kwargs['ha'] = 'center'

    if ('verticalalignment' not in kwargs) and ('va' not in kwargs):
        kwargs['va'] = 'center'

    if 'backgroundcolor' not in kwargs:
        kwargs['backgroundcolor'] = ax.get_facecolor()

    if 'clip_on' not in kwargs:
        kwargs['clip_on'] = True

    if 'zorder' not in kwargs:
        kwargs['zorder'] = 2.5

    print x,y,ax.get_xlim(),ax.get_ylim(), trans_angle
    if y > ax.get_ylim()[0] and y< ax.get_ylim()[1]:
        ax.text(x,y,label,rotation=trans_angle,**kwargs)


def labelLines(lines,align=True,xvals=None,**kwargs):

    ax = lines[0].axes
    labLines = []
    labels = []

    #Take only the lines which have labels other than the default ones
    for line in lines:
        label = line.get_label()
        if "_line" not in label:
            labLines.append(line)
            labels.append(label)

    if xvals is None:
        xmin,xmax = ax.get_xlim()
        print xmin, xmax
        xvals = np.linspace(xmin,xmax,len(labLines)+2)[1:-1]
        print xvals

    for line,x,label in zip(labLines,xvals,labels):
        labelLine(line,x,label,align,**kwargs)
```

D
Ressources internet

Liens valides le 16/09/2018.

D.1 SQL

L'aide en ligne est gigantesque. Je vous donne deux sites qui vous permettront d'obtenir les informations fondamentales

1. L'aide en ligne MariaDB en français : l'aide en ligne est gigantesque
2. Un site sur le SQL avec des cours, des liens etc... toujours en français ! https://sql.sh/

D.2 Python

Idem l'aide en ligne est gigantesque. Entrez votre question dans google 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/

D.3 Basemap et SIG

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

1. Le tutoriel en ligne, très bien fait : https://basemaptutorial.readthedocs.io/en/latest/
2. Quelques exemples de raster https://rabernat.github.io/research_computing/intro-to-basemap.html
3. Le cookbook ogr/gdal mais il n'est pas très friendly d'accès https://pcjericks.github.io/py-gdalogr-cookbook/
4. 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/
5. Des ressources intéressantes mais il faut chercher un peu. En français. http://www.portailsig.org/
6. Enfin une liste de librairies spatiales en python https://github.com/SpatialPython/spatial_python