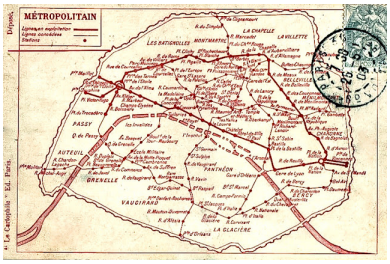
DS n° 03 – Métro parisien

- Faire tous les exercices dans un fichier NomPrenom.py à sauvegarder,
- mettre en commentaire l'exercice et la question traités (ex : # Exercice 1),
- ne pas oublier pas de commenter ce qui est fait dans votre code (ex : # Je crée une fonction pour calculer la racine d'un nombre),
- il est possible de demander un déblocage pour une question, mais celle-ci sera notée 0,
- il faut vérifier avant de partir que le code peut s'exécuter et qu'il affiche les résultats que vous attendez. Les lignes de code qui doivent s'exécuter sont décommentées.

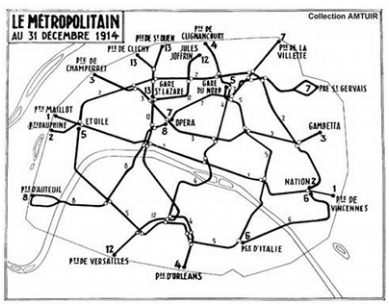
1 Introduction

Le plan du métro parisien a beaucoup évolué au cours du XXème siècle.

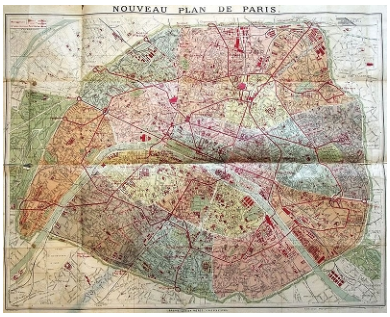
1903



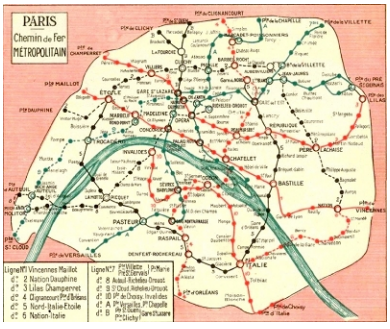
1914



1920



1930



1939

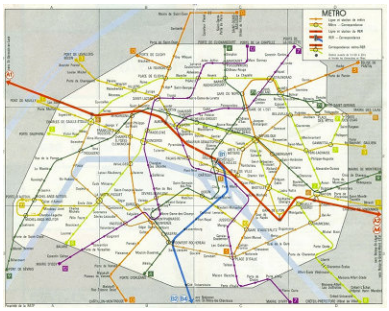
1950

1963

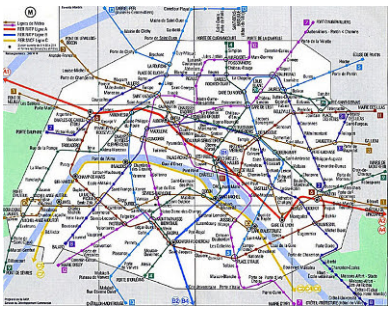
1967

1970

1978



1980

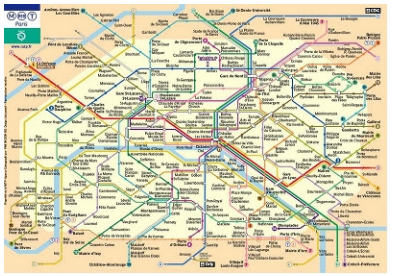


1991

1999

2000

2015



L'objectif de cette épreuve est de tracer le parcours d'une ligne de métro sur une carte à partir des données GPS des stations de métro de Paris.

Les données seront récupérées sur les deux fichiers csv dont le séparateur de colonne est le symbole ';' : `metro-paris-init.csv` et `metro-paris.csv`. Ces fichiers sont disponibles dans le répertoire « /home/eleve/Ressources/PTSI/ », ils ne doivent pas être déplacés mais ouverts directement depuis leur emplacement initial.

Les données ont été extraites d'une base contenant l'ensemble des stations de métro en France. Elle est disponible en suivant le lien suivant :
https://www.data.gouv.fr/fr/datasets/lignes-et-stations-de-metro-en-france/

2 Affichage de la carte

Afin de visualiser le tracé du métro sur une carte de Paris, nous allons grâce au code suivant importer une carte et l'afficher. La commande `plt.show()` trace la courbe et réinitialise les tracé, il faut, pour que la carte puisse être affichée dans la suite, la recharger avec la commande `plt.imshow(...)`, c'est pour cela que la ligne apparaît deux fois dans le code.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img=mpimg.imread('paris-map_b.png')
imgplot = plt.imshow(img, interpolation='none', aspect='auto')
plt.axis('equal')
plt.show()
imgplot = plt.imshow(img, interpolation='none', aspect='auto')
```

Question 1 **Recopier** le code ci-dessus et **l'exécuter** afin d'afficher la carte de Paris.

Le fichier `metro-paris-init.csv` contient les coordonnées de 4 stations « cardinales » du métro parisien qui sont situées à l'ouest (*La Défense (Grande Arche)*), au nord (*Porte de Clignancourt*), à l'est (*Château de Vincennes*) et au sud (*Porte d'Orléans*).

La positon de ces stations va permettre d'effectuer le calage entre les coordonnées GPS des stations et un équivalent pixels sur la carte image en png.

Pour cela, il faut commencer par lire le fichier `metro-paris-init.csv` et à partir de celui-ci générer une liste `stations_init` contenant la liste des 4 stations « cardinales », avec pour chaque station, son nom et une liste contenant sa latitude et sa longitude.

Ainsi, le premier des 4 éléments de la liste `stations_init` est alors :
`['"La Défense (Grande Arche)"', [48.89181786551305, 2.237988183098827]]`.

Question 2 **Proposer** un code permettant de lire le fichier `metro-paris-init.csv` et de générer la liste `stations_init`.

Le code suivant est fourni dans le fichier `code_fourni.py` disponible dans le dossier « /home/eleve/Ressources/PTSI/ » vous pouvez le copier/coller dans votre script.

```
# CODE FOURNI

# Coordonnées des stations en équivalent pixel sur la carte
coord_carte=[[197,187],[818,136.5],[1376.6,607.3],[706.6,792.2]]

# Calcul de l'échelle de la carte
rayon_terre=6361
distance_ns=rayon_terre*np.tan(np.pi*(stations_init[1][1][0]\
        -stations_init[0][1][0])/180.)
distance_eo=rayon_terre*np.tan(np.pi*(stations_init[1][1][1]\
        -stations_init[0][1][1])/180.)
echelle=[[distance_ns/(coord_carte[1][1]-coord_carte[0][1])],\
        [distance_eo/(coord_carte[1][0]-coord_carte[0][0])]]

# Fonction pour convertir les coordonnées GPS en équivalent pixel sur la carte image
def gps_to_map(coord_station):
    coord_px_x=rayon_terre*np.tan(np.pi*(coord_station[1]\
            -stations_init[0][1][1])/180.)/echelle[1]+coord_carte[0][0]
    coord_px_y=rayon_terre*np.tan(np.pi*(coord_station[0]\
            -stations_init[0][1][0])/180.)/echelle[0]+coord_carte[0][1]
    return coord_px_x,coord_px_y

# Tracé des stations sur la carte
for idx,station in enumerate(stations_init):
    coord_px_x,coord_px_y=gps_to_map(station[1])
    plt.scatter(coord_px_x,coord_px_y,s=30,c ='red')

# Tracé de la figure
plt.axis('equal')
```

```
plt.show()
imgplot = plt.imshow(img, interpolation='none', aspect='auto')
```

Question 3 **Copier/coller** le code fourni et l'exécuter. Si votre code fonctionne bien, la carte de Paris doit apparaître avec 4 points rouges au niveau des 4 stations « cardinales ».

Maintenant que la carte a été calée et que la fonction `gps_to_map(coord_station)` a été définie, il est possible de placer les stations sur la carte.

Question 4 En reprenant les codes des questions 2 et 3. **Proposer** un code permettant de générer la liste `stations` des stations de la ligne 1, au même format que la liste `stations_init` et **d'afficher** sur la carte l'ensemble des points correspondants à ces stations.

Question 5 **Modifier** le code de la question précédente afin de tracer la ligne reliant chacune des stations dans l'ordre dans lequel elle apparaît dans la liste. On rappelle que la fonction `plt.plot(x,y)` admet en entrée deux listes `x` et `y` contenant respectivement l'ensemble des coordonnées en x et en y de tous les points à relier.

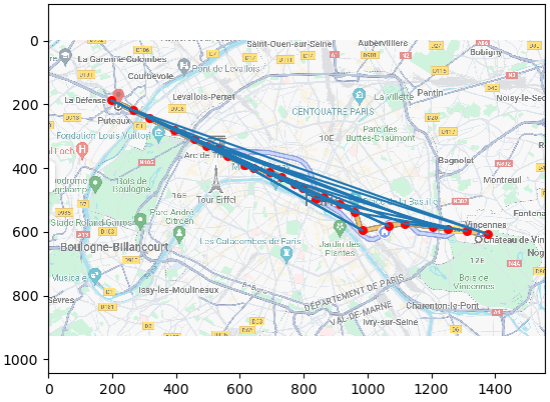


FIGURE 1 – Premier tracé du parcours de la ligne 1

La figure 1 montre le tracé de la ligne 1 obtenu grâce à la question 5. Le résultat n'est pas très probant.

3 Calcul de distance

Dans cette partie, l'objectif va être de déterminer la distance en km entre deux points $A[lati_A, long_A]$ et $B[lati_B, long_B]$ définis par leurs coordonnées GPS.

Pour cela on utilise la formule suivante :

$$distance = arccos(sin(\varphi_1) \times sin(\varphi_2) + cos(\varphi_1) \times cos(\varphi_2) \times cos(\lambda)) \times R$$

Avec :

$$\begin{cases} \varphi_1 = lati_A * \frac{\pi}{180} \\ \varphi_2 = lati_B * \frac{\pi}{180} \\ \lambda = (long_B - long_A) * \frac{\pi}{180} \\ R : Rayon\ de\ la\ terre \end{cases}$$

Question 6 **Coder** la fonction `distance(stationA,stationB)` qui renvoie la distance entre deux stations. Les stations en entrée de la fonction seront définies comme l'exemple suivant :

```
stationA=['"La Défense (Grande Arche)"',[48.891817865513, 2.2379881830988]]
```

Ainsi, la commande `print(distance(stations_init[0],stations_init[2]))` renvoie `15.704007162540035`.

4 Algorithme glouton

Afin de respecter l'ordre des stations d'une ligne durant le parcours, on va supposer que la station suivante est la plus proche. On verra que, selon les lignes, ce n'est pas toujours le cas.

Pour cela, nous allons générer une liste `liste_station_ordre` grâce à un algorithme glouton. Au départ, celle-ci ne contient que `stations[0]`.

L'algorithme à utiliser est le suivant :

— Partir de la dernière station de la liste `liste_station_ordre`,

— Chercher la station de la ligne 1 la plus proche (dont la distance est la plus petite) ET qui n'est pas déjà dans `liste_station_ordre`,

— Ajouter cette nouvelle station à `liste_station_ordre`,

— Recommencer à la première étape jusqu'à avoir récupéré toutes les stations de la ligne 1.

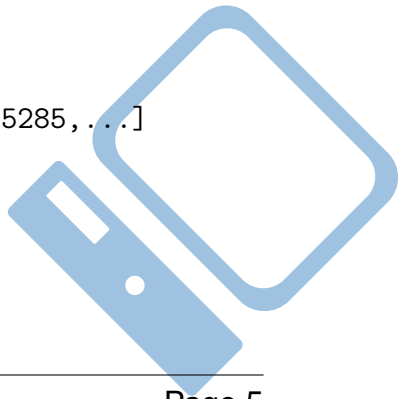
Question 7 **Coder** l'algorithme précédent afin de générer la liste `liste_station_ordre` puis parcourir cette liste afin de **tracer** les points et le parcours de la ligne sur la carte comme pour les questions 4 et 5.

Normalement, cette fois-ci le tracé est beaucoup plus probant.

Question 8 **Générer** et **afficher** la liste des distances parcourues cumulées, en mètre, au fur et à mesure du parcours de la ligne 1.

On obtient le résultat suivant :

```
distances=[0, 382, 1117, 1605, 2076, 2647, 3500, 4582, 5285,...]
```



5 Dichotomie

```
L=[1,3,5,7,10,13]
```

Question 9 Soit la liste `L` coder une fonction `dichotomie(L,a)` qui renvoie l'index d'une valeur dans la liste si celle-ci existe et `False` si la valeur n'est pas dans la liste.

Ex : `dichotomie(L,3)` renvoie `1` et `dichotomie(L,4)` renvoie `False`.

Question 10 Soit la liste `L` coder une fonction `dichotomie_interval` qui renvoie l'index d'une valeur dans la liste si celle-ci existe et les index des valeurs juste inférieure et juste supérieure si la valeur n'est pas dans la liste.

Ex : `dichotomie_interval(L,3)` renvoie `1` et `dichotomie_interval(L,4)` renvoie `(1,2)`

Question 11 Utiliser cette fonction pour déterminer combien de stations doivent être parcourues pour faire 10km sur la ligne 1.

6 Tracé de ligne

Question 12 **Coder** une fonction `trace_ligne(numero)` qui prend en entrée un numéro de ligne de métro et qui retourne le tracé de la ligne sur la carte. **Tester** la fonction avec la commande `trace_ligne(4)` .

FIN

Remarque : La suite du document peut et **doit** servir de brouillon.