TP 2 : Les villes de France

1 - Introduction

L'objectif de ce TP est d'aller plus loin dans l'utilisation de numpy en réalisant un programme qui permette de calculer des distances entre villes.

Le TP se décomposera en deux parties. Dans un premier temps, nous étudirons plusieurs approches pour calculer des distances avec les librairies `numpy` et `scipy` sur un exemple de synthèse. Puis dans un second temps, nous appliquerons ces principes pour calculer les distances entre les villes de Normandie.

Le TP sera à réaliser en python 3. Les librairies utilisées sont installées sur les machines de l'université, vous pouvez néanmoins les installer sur vos propres machines à l'aide de l'utilitaire pip présent par défaut avec python.

N'hésitez pas à regarder régulièrement la documentation de ces librairies, des exemples d'utilisation accompagnent généralement l'explication de chaque fonction.

- Python 3: https://docs.python.org/3/
- Numpy: https://docs.scipy.org/doc/numpy/reference/
- Scipy: https://docs.scipy.org/doc/scipy/reference/
- Matplotilb: https://matplotlib.org/contents.html

À part si cela est précisé, vous ne devez pas utiliser directement de boucle (f or , $whi \leq$) ou de branchement conditionnel (if) durant ce TP..

In [1]:
```
import numpy as np
import scipy as sc
import scipy.spatial
import matplotlib.pyplot as plt
```

Afin de vous guider dans la détection d'erreur dans votre code. Nous avons introduit des blocs de tests. Il n'est pas nécessaire que vous compreniez en détail le code de ces blocs. Vous devez uniquement les exécuter et corriger les erreurs de votre code si un des tests n'est pas valide. Il est important de noter que le fait de valider le test ne garantie pas que votre code ne contient pas d'erreur. Par contre un test non validé implique nécessairement que votre code contient une erreur.

- Si tout les tests sont valides, vous aurez un message écrit en vert indiquant : Ok - Tous les tests sont validés.
- Si un des tests n'est pas valide, vous aurez un message écrit en rouge indiquant : Au moins un test n'est pas validé.
- Pour les tests n'ont valide, vous aurez des éléments d'information sur le test non valide. En particulier, un message écrit en jaune vous détaillera la nature du test échoué.

Voici un exemple d'utilisation. Le bloc suivant est censé contenir l'affectation de la valeur 42 à

la variable `a`. Le bloc de test d'après vérifie que vous avez correctement effectué l'affectation. Exécutez les deux blocs avec des valeurs de `a` correcte et incorrecte.

In [2]:
```
a = 42
```

In [3]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(a,42,err_msg="\033[93m {}\033[00m" .format('Te
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

2 - Le calcul de distance avec `numpy` et `scipy`.

2.1 - Création de données de synthèses

Dans cette première section, nous créerons un ensemble de points pour tester ensuite différentes stratégies de calcul de distances entre ces points.

Créez un vecteur `x` de 10 valeurs entre 0 et 2π (2π compris).

In [4]:
```
x = np.linspace(0, 2 * np.pi, 10)
```

In [5]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(len(x),10,err_msg="\033[93m {}\033[00m" .forma
    np.testing.assert_equal(x[0],0,err_msg="\033[93m {}\033[00m" .format('
    np.testing.assert_equal(x[-1],2*np.pi,err_msg="\033[93m {}\033[00m" .f
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Concaténez les vecteurs $cos(x)$ et $sin(x)$ pour former un tableau `p` de taille (10,2).

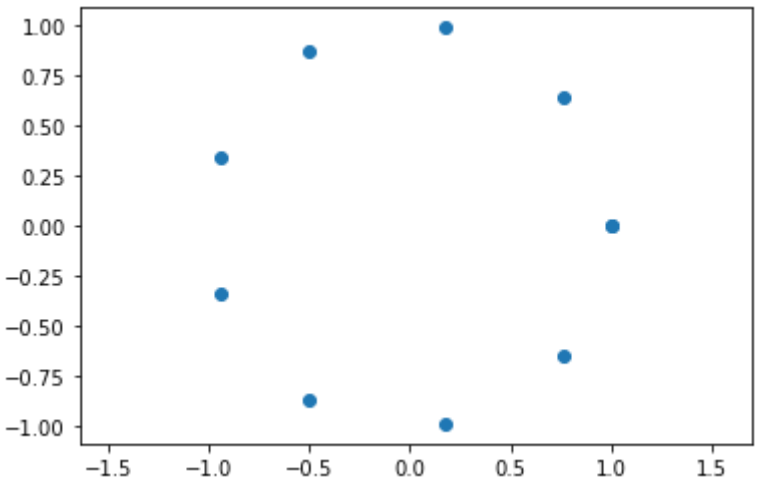
In [6]:
```
p = np.stack((np.cos(x), np.sin(x)), axis = 1)
```

In [7]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(p.shape,(10,2),err_msg="\033[93m {}\033[00m" .
    np.testing.assert_almost_equal(p[3,0],-0.5,err_msg="\033[93m {}\033[00
    np.testing.assert_almost_equal(p[3,1],np.sin(np.pi/3),err_msg="\033[93

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Affichez les 10 points que nous venons de créer avec la fonction `plt.scatter` . Vous pouvez ajouter `plt.axis('equal')` pour avoir des axes orthonormaux.

In [8]:
```
plt.scatter(p[:, 0], p[:, 1])
plt.axis("equal")
plt.show()
```



2.2 Calcul de distances euclidiennes entre points en utilisant seulement `numpy` .

Dans cette partie nous allons calculer la distance entre les 10 points se trouvant dans le tableau `p` en utilisant uniquement la librairie `numpy` . La solution que l'on va mettre en place n'est pas l'unique solution possible avec cette librairie (vous trouverez notament une autre solution dans le cours 2). A la fin de cette partie, nous aurons une matrice qui permet de retrouver la distance entre deux points en regardant la case associé aux 2 points. Par exemple la case de coordonnée (2,3) donne la distance entre le point `p[2]` et le point `p[3]` .

Créez deux versions de la matrice `p` de dimensions respectivement (1,10,2) et (10,1,2) que vous nommerez `p1` et `p2` .

In [9]:
```
p1 = p[np.newaxis, :] # ou bien p1 = np.reshape(p, (1,10,2))
p2 = p[:, np.newaxis] # ou bien p2 = np.reshape(p, (10, 1, 2))
```

In [10]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(p1.shape,(1,10,2),err_msg="\033[93m {}\033[00m
    np.testing.assert_equal(p2.shape,(10,1,2),err_msg="\033[93m {}\033[00m
    np.testing.assert_equal(p1[0,:,:],p2[:,0,:],err_msg="\033[93m {}\033[0(
    np.testing.assert_equal(p1.reshape(-1),p.reshape(-1),err_msg="\033[93m
    np.testing.assert_equal(p2.reshape(-1),p.reshape(-1),err_msg="\033[93m

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

En utilisant les deux tableaux `p1` et `p2` et le mécanisme de broadcasting, calculez la soustraction de toutes les valeurs de `p` entre elles. Le résultat se nommera `s` et sera de taille (10,10,2).

In [11]:
```
s = p1 - p2
s.shape
```

Out[11]: (10, 10, 2)

In [12]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(s.shape,(10,10,2),err_msg="\033[93m {}\033[00m
    np.testing.assert_almost_equal(s[3,4,:],
                                   [np.cos(2*np.pi*4/9)-np.cos(2*np.pi*3/9
                                   err_msg="\033[93m {}\033[00m" .format('
                                  )



except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Faites la somme des carrés des valeurs de `s` selon le dernier axe. Le résultat se nommera `dist2` .

In [13]:
```
dist2 = np.sum(s ** 2, axis = 2)
```

In [14]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(dist2.shape,(10,10),err_msg="\033[93m {}\033[0(
    np.testing.assert_almost_equal(s[7,2,0]**2+s[7,2,1]**2,
                                   dist2[7,2],
                                   err_msg="\033[93m {}\033[00m" .format('
                                  )



except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Calculez la racine carré des valeurs de `dist2` que vous mettrez dans la variable `dist_numpy` . Les cases de ce tableaux correspondent aux distances euclidiennes entre chacun des points.

In [15]:
```
dist_numpy = dist2 ** 0.5
```

In [16]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(dist_numpy.shape,(10,10),err_msg="\033[93m {}\(
    np.testing.assert_almost_equal(dist_numpy**2,dist2,
                                   err_msg="\033[93m {}\033[00m" .format('
                                  )



except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas 
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

En utilisant `dist_numpy` , quelle est la distance entre le point 3 et le point 5 ?

In [17]:
```
dist_numpy[3, 5]
```

Out[17]:
```
1.2855752193730787
```

2.3 Calcul de distances euclidiennes en utilisant `scipy` .

La fonction `sc.spatial.distance.pdist` de la librairie *scipy* (https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html) permet de calculer toutes les distances (par défaut la distance euclidienne) entre les points d'un ensemble de point. Elle

retourne ainsi un vecteur qui correspond d'abord à la distance du premier point avec tous les autres, puis la distance du deuxième point avec tous les autres sauf le premier point (en effet la distance entre le deuxième point et le premier est déjà calculée lorsque l'on a calculé la distance entre le premier point et le deuxième)... Cette fonction est optimisée pour calculer chaque distance une seule fois. Utilisez là pour calculer les distances des points `p` entre eux. Vous stockerez le résultat dans `md` .

In [18]:
```
md = sc.spatial.distance.pdist(p)
```

In [19]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(len(md),45,err_msg="\033[93m {}\033[00m" .forma
    np.testing.assert_almost_equal(md,[6.84040287e-01 ,1.28557522e+00 ,1.71
                                       err_msg="\033[93m {}\033[00m" .format('
                                      )



except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Nous allons maintenant réorganiser les valeurs pour avoir une matrice similaire à `dist_numpy` permettant de retrouver facilement la distance entre chaque point. Une case i, j de cette matrice, représente la distance entre le `p[i]` et le point `p[j]` .

Créez un vecteur `v` des entiers compris ente 0 et 10 (10 exclus).

In [20]:
```
v = np.arange(0,10)
```

In [21]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(len(v),10,err_msg="\033[93m {}\033[00m" .forma
    np.testing.assert_equal(v[0],0,err_msg="\033[93m {}\033[00m" .format('
    np.testing.assert_equal(v[-1],9,err_msg="\033[93m {}\033[00m" .format(
    np.testing.assert_equal(v[1:]-v[:-1],1,err_msg="\033[93m {}\033[00m" .

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Créez une matrice `m` contenant le vecteur `v` répété sur 10 lignes.

In [22]:
```
m = np.tile(v, (10,1))
#Ou bien on peut faire
# m = np.zeros((10,9)) + v
print('m=',m)
```

```
m= [[0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]
 [0 1 2 3 4 5 6 7 8 9]]
```

In [23]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(m.shape,(10,10),err_msg="\033[93m {}\033[00m"
    np.testing.assert_equal(np.sum(m,0),np.linspace(0,90,10),err_msg="\033
    np.testing.assert_equal(np.sum(m,1),45*np.ones(10),err_msg="\033[93m {

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Créez un tableau `t` de dimension (10,10,2) contenant une concaténation de la matrice m et de sa transposée. Il est plus simple de répondre à cette question avec la fonction `np.stack` .

In [24]:
```
t = np.stack((m, m.T), axis=2)
```

In [25]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(t.shape,(10,10,2),err_msg="\033[93m {}\033[00m
    np.testing.assert_equal(t[:,:,0],m,err_msg="\033[93m {}\033[00m" .forma
    np.testing.assert_equal(t[:,:,1],m.T,err_msg="\033[93m {}\033[00m" .fo

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

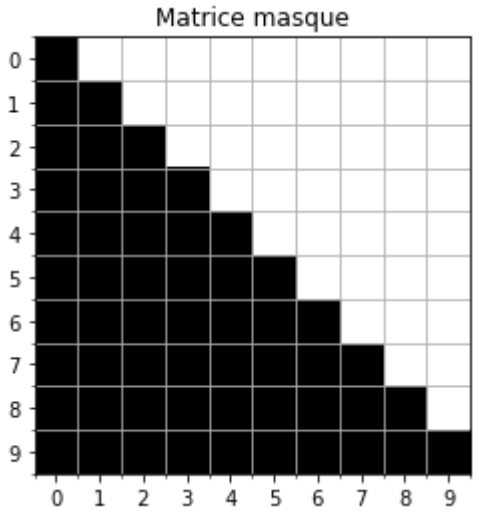
```
Ok - Tous les tests sont validés.
```

Lancez le code suivant, vous devriez avoir une matrice triangulaire supérieur. La visualisation de la matrice sous forme d'image peut être plus clair qu'un affichage direct de ses valeurs.

In [26]:

```
masque = 1*(t[:,:,0]>t[:,:,1])
print('Affichage direct des valeurs\n',masque)
plt.imshow(masque,cmap='gray')
plt.title('Matrice masque')
# Affichage de la grille
ax = plt.gca()
ax.set_xticks(np.arange(0, 10))
ax.set_yticks(np.arange(0, 10))
ax.set_xticks(np.arange(-.5, 10, 1), minor=True);
ax.set_yticks(np.arange(-.5, 10, 1), minor=True);
ax.set_xticklabels(np.arange(10))
ax.set_yticklabels(np.arange(10))
plt.grid(which='minor')
plt.show()
```

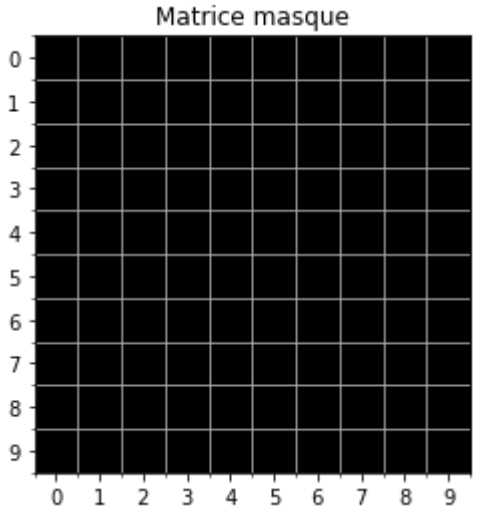
```
Affichage direct des valeurs
 [[0 1 1 1 1 1 1 1 1 1]
 [0 0 1 1 1 1 1 1 1 1]
 [0 0 0 1 1 1 1 1 1 1]
 [0 0 0 0 1 1 1 1 1 1]
 [0 0 0 0 0 1 1 1 1 1]
 [0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 1 1 1]
 [0 0 0 0 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0]]
```



Nous voulons créez une matrice permettant de connaitre les distances entre les 10 points étudiés. Une case i, j représente la distance entre le point i et le point j. Commencez par créer une matrice `dist` de (10,10) que vous initialiserez à 0.

In [27]:
```
dist = np.zeros((10,10))
print(dist)
plt.imshow(dist,cmap='gray')
plt.title('Matrice masque')
# Affichage de la grille
ax = plt.gca()
ax.set_xticks(np.arange(0, 10))
ax.set_yticks(np.arange(0, 10))
ax.set_xticks(np.arange(-.5, 10, 1), minor=True);
ax.set_yticks(np.arange(-.5, 10, 1), minor=True);
ax.set_xticklabels(np.arange(10))
ax.set_yticklabels(np.arange(10))
plt.grid(which='minor')
plt.show()
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```



La matrice `masque` calculée précédemment indique entre quels points sont calculés les distances du vecteur `md` (calculé par la fonction `sc.spatial.distance.pdist`). Les 1 de la matrice `masque` sont les endroits où devront être recopié les valeurs du vecteur `md` .

Vérifiez qu'il y a autant d'éléments dans le vecteur `md` que de 1 dans la matrice `masque` .

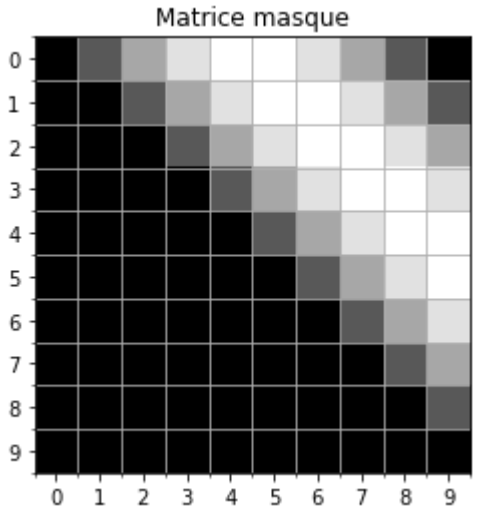
In [28]:
```
print("Longueur du vecteur md: {}\n Nombre de 1 dans masque: {}".format(le
```

```
Longueur du vecteur md: 45
 Nombre de 1 dans masque: 45
```

Recopiez les valeurs de `md` dans `dist` aux positions où le `masque==1` . Lors de la recopie, l'ordre de parcours correspond à l'ordre normal sur les tableaux numpy.

In [29]:

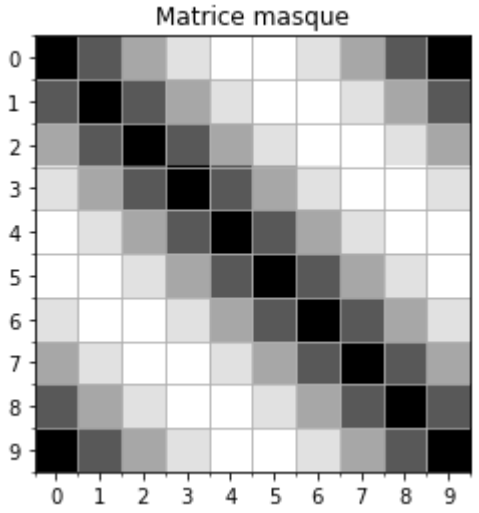
```
dist[masque == 1] = md
plt.imshow(dist,cmap='gray')
plt.title('Matrice masque')
# Affichage de la grille
ax = plt.gca()
ax.set_xticks(np.arange(0, 10))
ax.set_yticks(np.arange(0, 10))
ax.set_xticks(np.arange(-.5, 10, 1), minor=True);
ax.set_yticks(np.arange(-.5, 10, 1), minor=True);
ax.set_xticklabels(np.arange(10))
ax.set_yticklabels(np.arange(10))
plt.grid(which='minor')
plt.show()
```



Il faut maintenant compléter la matrice `dist` avec son symétrique pour avoir la matrice de toutes les distances. La diagonale peut être laissé à 0 (la distance d'un point avec lui même est bien nulle).

In [30]:

```
dist += dist.T
plt.imshow(dist,cmap='gray')
plt.title('Matrice masque')
# Affichage de la grille
ax = plt.gca()
ax.set_xticks(np.arange(0, 10))
ax.set_yticks(np.arange(0, 10))
ax.set_xticks(np.arange(-.5, 10, 1), minor=True);
ax.set_yticks(np.arange(-.5, 10, 1), minor=True);
ax.set_xticklabels(np.arange(10))
ax.set_yticklabels(np.arange(10))
plt.grid(which='minor')
plt.show()
```



Vous pouvez vérifiez la distance entre le premier et le cinquième point.

In [31]:
```
print("Distance entre le point 0 et le point 4 : {}\nDistance entre le poi
```

```
Distance entre le point 0 et le point 4 : 1.969615506024416
Distance entre le point 4 et le point 0 : 1.969615506024416
```

L'opération que nous venons de faire peut être faite directement avec la fonction `sc.spatial.distance.squareform` de scipy (https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.squareform.html#scipy.spatial.distance.squareform). Utilisez cette fonction sur `md` et comparez le résultat avec la matrice `dist` et la matrice `dist_numpy` .

In [32]:
```
dist_avec_scipy = sc.spatial.distance.squareform(md)

print("Egalité entre les 3 distances {}".format((np.all(dist_avec_scipy==di
```

```
Egalité entre les 3 distances True
```

3 - Application du calcul des distances sur les villes de Normandie

3.1 Analyse du fichier fourni

Nous allons travailler sur une version modifiée d'un fichier recensant l'ensemble des villes de France dont la version originale se trouve à l'adresse suivante: https://www.data.gouv.fr/fr/datasets/listes-des-communes-geolocalisees-par-regions-departements-circonscriptions-nd/. Vous avez à votre disposition deux fichiers appelé *villes_france.csv* et *villes_normandie.csv* contenant des informations sur respectivement l'ensemble des villes de France et de Normandie. Vous pouvez commencer à regarder leurs contenus au moyen d'un éditeur de texte (par exemple gedit), ou l'importer sous Open Office ou Libre Office.

La commande shell (linux) suivante permet aussi de visualiser les premières lignes du fichier et donne un aperçu de sa structure.

- Les colonnes sont séparées par des ";".
- La première ligne décrit les intitulés des différentes colonnes.

- Chaque ligne suivante recense une ville donnée.

Rq: le point d'exclamation permet d'exécuter des commandes shell dans un notebook jupyter.

In [33]:
```
!head -n 6 villes_normandie.csv
```

```
EU_circo;code_région;nom_région;chef-lieu_région;numéro_département;nom_dép
artement;préfecture;numéro_circonscription;nom_commune;codes_postaux;code_i
nsee;latitude;longitude;éloignement
Nord-Ouest;25;Basse-Normandie;Caen;14;Calvados;Caen;1;Audrieu;14250;14026;4
9.2;-0.6;0.75
Nord-Ouest;25;Basse-Normandie;Caen;14;Calvados;Caen;1;Authie;14280;14030;4
9.2;-0.433333;1.07
Nord-Ouest;25;Basse-Normandie;Caen;14;Calvados;Caen;1;Bretteville-sur-Odon;
14760;14101;49.166667;-0.416667;0.37
Nord-Ouest;25;Basse-Normandie;Caen;14;Calvados;Caen;1;Brouay;14250;14109;4
9.216667;-0.566667;1.52
Nord-Ouest;25;Basse-Normandie;Caen;14;Calvados;Caen;1;Caen;14000;14118;49.1
83333;-0.35;0.5
```

Vous travaillerez dans un premier sur le fichier *villes_normandie.csv*. Une fois votre code fonctionnel, vous pouvez tester sur l'ensemble des villes de France *villes_france.csv*.

Le code suivant importe le contenu du fichier *villes_normandie.csv* dans la variable `nom_ville`. Seule la colonne 8 est importé, chaque colonne étant défini grâce au séparateur `;`.

In [34]:
```
nom_ville = np.loadtxt('villes_normandie.csv', delimiter=';',dtype=np.unic
print(nom_ville)
```

```
['Audrieu' 'Authie' 'Bretteville-sur-Odon' ... 'Vieux-Rouen-sur-Bresle'
 'Villers-sous-Foucarmont' 'Wanchy-Capval']
<ipython-input-34-6358a5e47513>:1: DeprecationWarning: `np.unicode` is a de
precated alias for `np.compat.unicode`. To silence this warning, use `np.co
mpat.unicode` by itself. In the likely event your code does not need to wor
k on Python 2 you can use the builtin `str` for which `np.compat.unicode` i
s itself an alias. Doing this will not modify any behaviour and is safe. If
you specifically wanted the numpy scalar type, use `np.str_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  nom_ville = np.loadtxt('villes_normandie.csv', delimiter=';',dtype=np.uni
code,usecols=8,skiprows=1)
```

Le code suivant permet de détecter les villes portant le même nom. Nous les supprimerons dans la suite du TP. La variable `premier_doublon` indique l'indice de chaque première occurrence de nom des villes.

In [35]:
```
# Détection des doublons
nom_ville, premier_doublon = np.unique(nom_ville, return_index=True)
print(nom_ville)
```

```
['Ablon' 'Aclou' 'Acon' ... 'Yville-sur-Seine' 'Yvrandes' 'Yébleron']
```

En utilisant le tableau `nom_ville` et une boucle `for`, créez un dictionnaire python nommé `ville` donnant pour un nom de ville son numéro correspondant à son ordre d'apparition dans le tableau `nom_ville`. Par exemple *Ablon* aura le numéro 0, *Caen* le numéro 428... Vous convertirez les chaines de caractères numpy en chaine de caractères python comme dans l'affichage ci-dessus.

In [36]:
```
ville = {k: v for (k, v) in zip(nom_ville, range(len(nom_ville)))}
```

In [37]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(type(ville),type({}),err_msg="\033[93m {}\033[(
    np.testing.assert_equal(nom_ville[ville['Audrieu']],'Audrieu',err_msg=
    np.testing.assert_equal(nom_ville[ville['Caen']],'Caen',err_msg="\033[9
    np.testing.assert_equal(len(ville),2946,err_msg="\033[93m {}\033[00m"

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Créez une matrice numpy `coord` dans laquelle chaque ligne représente une ville, la première colonne contient la latitude et la seconde colonne la longitude de la ville. Pour cela remplacez dans le code suivant les "?" avec les bonnes valeurs. Vous aurez besoin d'ouvrir le fichier *villes_normandie.csv* avec un éditeur de texte pour connaitre les indices des colonnes à récupérer.

In [38]:
```
coord = np.loadtxt('villes_normandie.csv', delimiter=';',usecols=(11,12),sk
coord = coord[premier_doublon]
```

In [39]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(coord[428],[49.183333,-0.35 ],err_msg="\033[93m
    np.testing.assert_equal(coord[ville['Caen']],[49.183333,-0.35 ],err_msg

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Sauver le dictionnaire et les matrices dans un fichier de type "pickle", cela sera plus facile à charger ultérieurement.

In [40]:
```
import pickle

# pour écrire les données sur disque avec pickle
with open('data.pickle', 'wb') as f:
    pickle.dump([ville,nom_ville,coord], f)

# pour lire les données sur disque avec pickle
#with open('data.pickle', 'rb') as f:
#    [ville,nom_ville,coord] = pickle.load(f)
```

3.2 - Distances géodésiques entre villes

Dans un premier temps, vous écrirez une fonction `distGeo` qui permet de calculer la distance géodésique entre deux villes, à partir des coordonnées des villes. Vous utiliserez pour cela l'équation suivante :

$$d_g = R \cos^{-1}(\sin(p1_{lat}) \sin(p2_{lat}) + \cos(p1_{lat}) \cos(p2_{lat}) \cos(p1_{lon} - p2_{lon}))$$

où $R = 6367.445$ est le rayon de la terre (en km), $p1$ et $p2$ les deux points dont on souhaite calculer la distance. Les indices $_{lat}$ $_{lon}$ représentent la latitude et la longitude en radian. Attention, dans le ficher de données, les valeurs sont données en degrés.

In [41]:
```
R = 6367.445

def distGeo(u,v):
    u1 = np.deg2rad(u)
    v1 = np.deg2rad(v)
    return R * np.arccos(np.sin(u1[0]) * np.sin(v1[0]) + np.cos(u1[0]) * np
```

In [42]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_almost_equal(distGeo(np.array([43,-88]),np.array([43
    np.testing.assert_almost_equal(distGeo(np.array([10,-30])+360,np.array
    np.testing.assert_almost_equal(distGeo(np.array([10,-30]),360+np.array
    np.testing.assert_almost_equal(distGeo(coord[ville["Caen"]],coord[ville

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Vérifiez que votre fonction donne un résultat correct en calculant la distance entre Caen et Rouen et comparez le résultat à celui donné par le site https://www.ephemeride.com/atlas/distance/27/ qui permet de calculer les distances à vol d'oiseau entre deux villes.

In [43]:
```
print("Distance entre Caen et Rouen: {}".format(distGeo(coord[ville["Caen"]
```

```
Distance entre Caen et Rouen: 107.5056062486595
```

Nous désirons désormais calculer une matrice `dist` contenant l'ensemble des distances entre villes. La valeur (i, j) de cette matrice représentera la distance entre la ville i et la ville j.

Construire cette matrice avec des boucles prendrait beaucoup trop de temps. Utilisez les fonctions `pdist` et `squareform` vu en début de TP. Nous attirons votre attention sur l'argument `metric` de la fonction `pdist` permettant de choisir le type de distance utilisée. Cet argument peut être soit une chaîne de caractère parmi celles proposées dans la documentation, soit une fonction que vous avez préalablement créée (comme ici la fonction `distGeo`).

Calculez cette matrice de distance pour les 500 premières villes (matrice 500x500).
Affichez également le temps d'exécution de ce bloc d'instruction.

In [44]:
```
%%time
md = sc.spatial.distance.pdist(coord[:500], metric = distGeo)
dist = sc.spatial.distance.squareform(md)
```

```
CPU times: user 1.03 s, sys: 2.08 ms, total: 1.03 s
Wall time: 1.03 s
```

In [45]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_almost_equal(dist[ville['Caen'],ville['Audrieu']],18

except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

```
Ok - Tous les tests sont validés.
```

Le calcul de l'ensemble des distances entre les villes est une opération de complexité quadratique. Cela veux dire que si le nombre de ville augmente d'un facteur n alors le temps de calcul augmentera à peu près d'un facteur n^2. À votre avis que va-t-il se passer si on veut calculer cette matrice pour l'ensemble des villes de Normandie ? Pour l'ensemble des villes de France ?

In [46]:
```
%%time
md2 = sc.spatial.distance.pdist(coord, metric = distGeo)
dist2 = sc.spatial.distance.squareform(md2)
```

```
CPU times: user 36.1 s, sys: 15.8 ms, total: 36.2 s
Wall time: 36.2 s
```

Votre réponse: Le calcul de la distance entre toute les villes a augmenté avec un facteur d'environs qui est compris entre [30.5, 36] Le nombre de villes a augmenté d'un facteur de [5.5, 6] Les résultats pratiques corrobores l'hypothèse

3.3- Distances euclidiennes entre villes

Nous allons à présent mettre en place une méthode plus rapide pour calculer la distance entre les différentes villes. Nous allons commencer par représenter les villes par leurs coordonnées 3D, et c'est à partir de ces coordonnées 3D que l'on calculera les distances euclidiennes entre villes. La distance euclidienne est moins précise que la distance géodésique car elle ne prend pas en compte la courbure de la terre mais son calcul est plus rapide.

Pour calculer les coordonnées 3D, vous pourrez utiliser les équations suivantes :

$$x = R\ \cos(p_{lat})\sin(p_{lon})$$
$$y = R\ \cos(p_{lat})\cos(p_{lon})$$
$$z = R\ \sin(p_{lat})$$

Attention, les angles doivent être en radian et non en degré et $R = 6367.445$ est le rayon de la terre.

Écrivez une fonction `convert3D` qui convertit un tableau de coordonnées décrites en terme de latitude et de longitude en un tableau de coordonnées 3D.

In [47]:
```
def convert3D(u):
    xyz = np.zeros((u.shape[0],3))
    u1 = np.deg2rad(u)
    xyz[:,0] = R * np.cos(u1[:,0]) * np.sin(u1[:,1])
    xyz[:,1] = R * np.cos(u1[:,0]) * np.cos(u1[:,1])
    xyz[:,2] = R * np.sin(u1[:,0])
    return xyz
```

In [48]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_almost_equal(convert3D(np.array([[30,-47],[78,44]]))
                                   (2,3),
                                   err_msg="\033[93m {}\033[00m" .format('
                                  )
    np.testing.assert_almost_equal(convert3D(np.array([[30,-47],[78,44]]))
                                   [-4032.95427327 , 919.6347762 ],
                                   err_msg="\033[93m {}\033[00m" .format('
                                  )
    np.testing.assert_almost_equal(convert3D(np.array([[30,-47],[78,44]]))
                                   [3760.79070153,  952.30968837],
                                   err_msg="\033[93m {}\033[00m" .format('
                                  )
    np.testing.assert_almost_equal(convert3D(np.array([[30,-47],[78,44]]))
                                   [3183.7225 ,    6228.30104955],
                                   err_msg="\033[93m {}\033[00m" .format('
                                  )
except Exception as e:
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)

else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

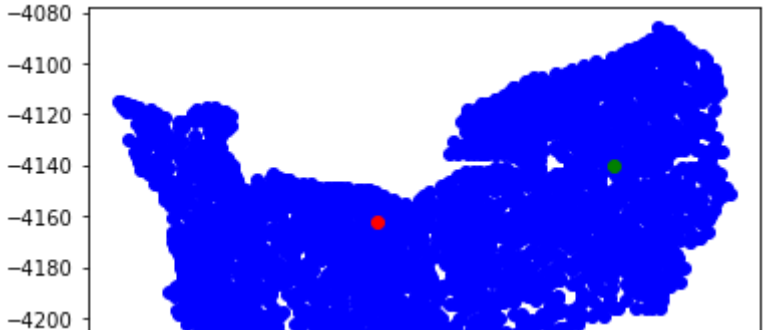
```
Ok - Tous les tests sont validés.
```

En utilisant la fonction `plt.scatter` de *matplotlib*, affichez chaque ville de Normandie sous la forme de point de coordonnées $(x, -y)$. On ignorera la 3ème coordonnée z qui correspond à la distance avec le centre de la terre.

Vous pouvez afficher d'une couleur différente les villes de *Caen* et *Rouen*.

In [49]:
```
coord1 = convert3D(coord)[:, :2]
plt.scatter(coord1[:, 0], -coord1[:, 1], color="blue")
plt.scatter(coord1[ville["Caen"]][0], - coord1[ville["Caen"]][1], color="r
plt.scatter(coord1[ville["Rouen"]][0], - coord1[ville["Rouen"]][1], color=
```

Out[49]:
```
<matplotlib.collections.PathCollection at 0x7f3054c62fa0>
```



Créez une fonction `distEuc` qui calcul la distance euclidienne entre deux villes dont les coordonnées sont en latitude, longitude. Vous utiliserez une fonction de concaténation et la fonction `convert3D` pour faire cette fonction.

In [50]:
```
def distEuc(u,v):
    p = np.stack((u,v), axis=0)
    p = convert3D(p)[:, :2]
    return np.sqrt((p[0, 0] - p[1, 0]) ** 2 + (p[0, 1] - p[1, 1]) ** 2)
```

Calculez la distance entre *Caen* et *Rouen* en utilisant `distEuc` . Trouvez-vous le même résultat qu'avec une distance géodésique ?

In [51]:
```
md3 = sc.spatial.distance.pdist(coord, metric=distEuc)
dist3 = sc.spatial.distance.squareform(md3)
print("Distance ente Caen et Rouen: {}".format(dist3[ville["Caen"], ville[
#La distance trouvé est differente de celle de la distance geodessique qui
```

```
Distance ente Caen et Rouen: 105.96722327418952
```