TP 3 : Propagation d'une maladie en Normandie

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 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

import pickle

# Pour les tests
import sys
from io import StringIO
```

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 garantit 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 non valides, 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. Vous devez supprimer la ligne `raise NotImplementedError()` qui indique que vous n'avez pas encore fait l'implémentation.

In [2]:
```
# YOUR CODE HERE
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

2 - Villes les plus proches / Villes les plus loins

Nous allons écrire un programme qui recherche les 10 villes les plus proches d'une ville donnée. Commencez par récupérer les coordonnées des villes de Normandie sans doublon que nous avions calculées dans le TP 2. Les valeurs avait été placées dans un fichier `data.pickle`. Vous placerez les coordonnées dans la variable `coord`.

In [4]:
```
#pour lire les données sur disque avec pickle
with open('data.pickle', 'rb') as f:
    [ville,nom_ville,coord] = pickle.load(f)
```

In [5]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(coord.shape,(2946,2),err_msg="\033[93m {}\033[(
    np.testing.assert_almost_equal(coord[0],[49.4,0.3],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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

In [6]:
```
R = 6367.445
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

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

ville = {key: value for key, value in zip(nom_ville, range(len(nom_ville))
```

Calculez la distance entre toutes les villes de Normandie (sans doublon) que vous placerez dans la variable `d_villes` . Pour cela nous allons utiliser les fonctions `pdist` et `squareform` comme au TP 2 et une distance euclidienne sur les points 3D.
Attention les coordonnées doivent être sous la forme x,y,z et non latitude/longitude.

In [7]:
```
%%time
md = sc.spatial.distance.pdist(convert3D(coord))
d_villes = sc.spatial.distance.squareform(md)
```

```
CPU times: user 31.1 ms, sys: 36.8 ms, total: 67.9 ms
Wall time: 66.5 ms
```

In [8]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(d_villes.shape,(2946,2946),err_msg="\033[93m {}
    np.testing.assert_almost_equal(d_villes-d_villes.T,np.zeros(d_villes.sh
    np.testing.assert_almost_equal(np.sum(d_villes),888370473.7872453,err_m
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

Récupérez, dans la variable `dist_Caen` , la distance entre *Caen* et les autres villes de Normandie. Il s'agit de retrouver dans la matrice `d_villes` la ligne associée à *Caen*.

In [9]:
```
# YOUR CODE HERE
dist_Caen = d_villes[ville["Caen"]]
```

In [10]:

```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(dist_Caen.shape,(2946,),err_msg="\033[93m {}\0
    np.testing.assert_almost_equal(dist_Caen[:3],[52.9093495,  76.3072865,
    np.testing.assert_almost_equal(np.sum(dist_Caen),237420.0423234083,err_
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

En utilisant le tableau `nom_ville` défini au TP 2 et la fonction `np.argsort` qui permet d'ordonner les indices du vecteur `dist_Caen` selon l'ordre croissant des distances associées, donnez le nom des 10 villes les plus proches de *Caen*.

Pour cela vous compléterez la fonction `affichage_sans_mise_en_forme_10procheCaen` . Cette fonction ne retourne rien et fait l'affichage du résultat. Le rendu doit être similaire à :

```
['Hérouville-Saint-Clair' 'Louvigny' 'Mondeville'
'Saint-Germain-la-Blanche-Herbe' 'Epron' 'Colombelles'
'Fleury-sur-Orne'
'Giberville' 'Bretteville-sur-Odon' 'Saint-Contest']
```

In [11]:

```
def affichage_sans_mise_en_forme_10procheCaen():
    # YOUR CODE HERE
    global dist_Caen, nom_ville
    indice = np.argsort(dist_Caen)[1:11]
    nom_villes = nom_ville[indice]
    print(nom_villes)

print('\033[34mAffichage sans mise en forme:\033[0m\n')
affichage_sans_mise_en_forme_10procheCaen()
```

```
Affichage sans mise en forme:

['Hérouville-Saint-Clair' 'Louvigny' 'Mondeville'
 'Saint-Germain-la-Blanche-Herbe' 'Epron' 'Colombelles' 'Fleury-sur-Orne'
 'Giberville' 'Bretteville-sur-Odon' 'Saint-Contest']
```

In [12]:

```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
_str_print_ = StringIO()
_original_stdout = sys.stdout
sys.stdout = _str_print_

affichage_sans_mise_en_forme_10procheCaen()
try:
    np.testing.assert_equal(_str_print_.getvalue().replace('\n',''),
                            "['Hérouville-Saint-Clair' 'Louvigny' 'Mondevi
                           err_msg="\033[93m {}\033[00m" .format('Test 1 :
                           )
except Exception as e:
    sys.stdout = _original_stdout
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    sys.stdout = _original_stdout
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

En utilisant une boucle `for` mettez en forme l'affichage pour avoir le rendu suivant:

```
Hérouville-Saint-Clair est à une distance de 3.05km de Caen.
Louvigny est à une distance de 3.05km de Caen.
Mondeville est à une distance de 3.05km de Caen.
Saint-Germain-la-Blanche-Herbe est à une distance de 3.63km
de Caen.
Epron est à une distance de 3.9km de Caen.
Colombelles est à une distance de 4.08km de Caen.
Fleury-sur-Orne est à une distance de 4.43km de Caen.
Giberville est à une distance de 4.84km de Caen.
Bretteville-sur-Odon est à une distance de 5.19km de Caen.
Saint-Contest est à une distance de 5.19km de Caen.
```

Vous utiliserez la fonction `np.round` pour les arrondis.

In [13]:

```
def affichage_avec_mise_en_forme_10procheCaen():
    # YOUR CODE HERE
    global dist_Caen, nom_ville
    indice = np.argsort(dist_Caen)[1:11]
    dist_proche = dist_Caen[indice]
    nom_villes = nom_ville[indice]
    for nom, dist in zip(nom_villes, dist_proche):
        print(f"{nom} est à une distance de {np.round(dist, 2)}km de Caen.

print('\033[34mAffichage avec mise en forme: \033[0m')
print()
affichage_avec_mise_en_forme_10procheCaen()
```

```
Affichage avec mise en forme:

Hérouville-Saint-Clair est à une distance de 3.05km de Caen.
Louvigny est à une distance de 3.05km de Caen.
Mondeville est à une distance de 3.05km de Caen.
```

```
Saint-Germain-la-Blanche-Herbe est à une distance de 3.63km de Caen.
Epron est à une distance de 3.9km de Caen.
Colombelles est à une distance de 4.08km de Caen.
Fleury-sur-Orne est à une distance de 4.43km de Caen.
Giberville est à une distance de 4.84km de Caen.
Bretteville-sur-Odon est à une distance de 5.19km de Caen.
Saint-Contest est à une distance de 5.19km de Caen.
```

In [14]:

```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
_str_print_ = StringIO()
_original_stdout = sys.stdout
sys.stdout = _str_print_

affichage_avec_mise_en_forme_10procheCaen()
try:
    np.testing.assert_equal(_str_print_.getvalue(),
                            "Hérouville-Saint-Clair est à une distance de 1
Louvigny est à une distance de 3.05km de Caen.\n\
Mondeville est à une distance de 3.05km de Caen.\n\
Saint-Germain-la-Blanche-Herbe est à une distance de 3.63km de Caen.\n\
Epron est à une distance de 3.9km de Caen.\n\
Colombelles est à une distance de 4.08km de Caen.\n\
Fleury-sur-Orne est à une distance de 4.43km de Caen.\n\
Giberville est à une distance de 4.84km de Caen.\n\
Bretteville-sur-Odon est à une distance de 5.19km de Caen.\n\
Saint-Contest est à une distance de 5.19km de Caen.\n",
                           err_msg="\033[93m {}\033[00m" .format('Test 1 :
                           )
except Exception as e:
    sys.stdout = _original_stdout
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    sys.stdout = _original_stdout
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

In [15]:

```
dist_Rouen = d_villes[ville["Rouen"]]
```

Vous ferez de même avec le calcul des 10 villes les plus éloignées de *Rouen*.

In [16]:

```
def affichage_sans_mise_en_forme_10loinRouen():
    # YOUR CODE HERE
    global dist_Rouen, nom_ville
    indice = np.argsort(dist_Rouen)[-1:-11:-1]
    nom_villes = nom_ville[indice]
    print(nom_villes)


def affichage_avec_mise_en_forme_10loinRouen():
    # YOUR CODE HERE
    global dist_Rouen, nom_ville
    indice = np.argsort(dist_Rouen)[-1:-11:-1]
    dist_proche = dist_Rouen[indice]
    nom_villes = nom_ville[indice]
    for nom, dist in zip(nom_villes, dist_proche):
        print(f"{nom} est à une distance de {np.round(dist, 2)}km de Rouen

print('\033[34mAffichage sans mise en forme:\033[0m')
affichage_sans_mise_en_forme_10loinRouen()

print()
print('\033[34mAffichage avec mise en forme: \033[0m')
affichage_avec_mise_en_forme_10loinRouen()
```

```
Affichage sans mise en forme:
['Auderville' 'Saint-Germain-des-Vaux' 'Jobourg' 'Omonville-la-Petite'
 'Herqueville' 'Digulleville' 'Pontorson' 'Omonville-la-Rogue'
 'Flamanville' 'Beaumont-Hague']

Affichage avec mise en forme:
Auderville est à une distance de 219.43km de Rouen.
Saint-Germain-des-Vaux est à une distance de 218.67km de Rouen.
Jobourg est à une distance de 217.1km de Rouen.
Omonville-la-Petite est à une distance de 215.85km de Rouen.
Herqueville est à une distance de 215.44km de Rouen.
Digulleville est à une distance de 214.66km de Rouen.
Pontorson est à une distance de 213.48km de Rouen.
Omonville-la-Rogue est à une distance de 213.47km de Rouen.
Flamanville est à une distance de 213.25km de Rouen.
Beaumont-Hague est à une distance de 213.06km de Rouen.
```

In [17]:

```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
_str_print_ = StringIO()
_original_stdout = sys.stdout
sys.stdout = _str_print_

try:
    affichage_sans_mise_en_forme_10loinRouen()
    np.testing.assert_equal(_str_print_.getvalue().replace('\n',''),
                            "['Auderville' 'Saint-Germain-des-Vaux' 'Jobou
                           err_msg="\033[93m {}\033[00m" .format('Test 1 :
                           )
    _str_print_.truncate(0)
    _str_print_.seek(0)
    affichage_avec_mise_en_forme_10loinRouen()
    np.testing.assert_equal(_str_print_.getvalue(),
                            "Auderville est à une distance de 219.43km de R
Saint-Germain-des-Vaux est à une distance de 218.67km de Rouen.\n\
Jobourg est à une distance de 217.1km de Rouen.\n\
Omonville-la-Petite est à une distance de 215.85km de Rouen.\n\
Herqueville est à une distance de 215.44km de Rouen.\n\
Digulleville est à une distance de 214.66km de Rouen.\n\
Pontorson est à une distance de 213.48km de Rouen.\n\
Omonville-la-Rogue est à une distance de 213.47km de Rouen.\n\
Flamanville est à une distance de 213.25km de Rouen.\n\
Beaumont-Hague est à une distance de 213.06km de Rouen.\n",
                           err_msg="\033[93m {}\033[00m" .format('Test 2 :
                           )
except Exception as e:
    sys.stdout = _original_stdout
    print("\033[91m {}\033[00m" .format('KO - Au moins un test n\'est pas v
    print('Information sur le test non valide:')
    print(e)
    raise e
else:
    sys.stdout = _original_stdout
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

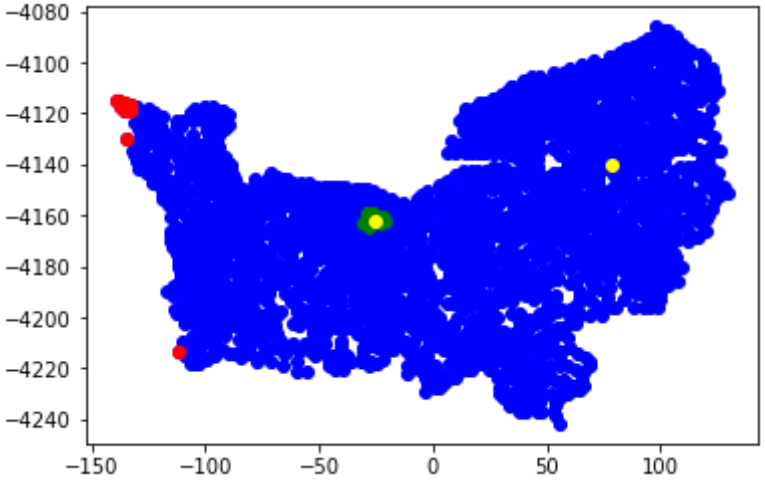
```
Ok - Tous les tests sont validés.
```

Affichez les villes de Normandie dans une figure matplotlib et mettez :

- en jaune, *Caen* et *Rouen*,
- en vert, les 10 villes les plus proches de *Caen*,
- en rouge, les 10 villes les plus loin de *Rouen*.

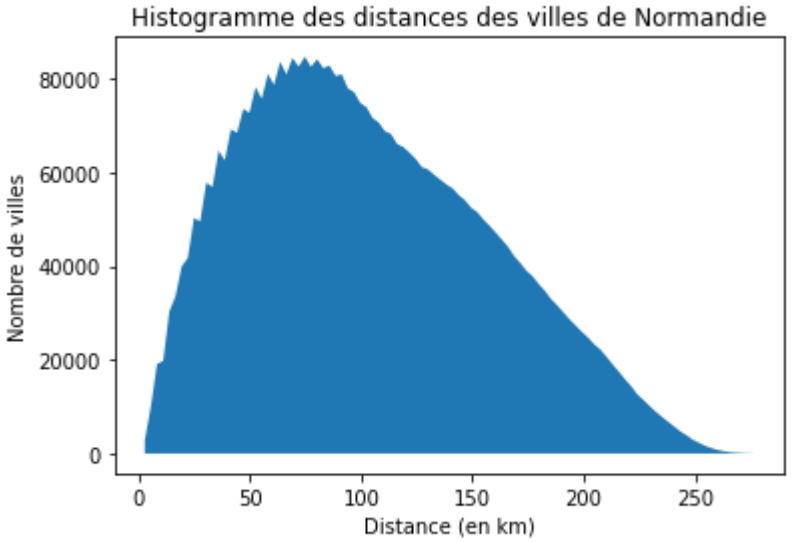
Le rendu devrait être similaire à :

In [18]:
```
# YOUR CODE HERE
indice_proche_caen = np.argsort(dist_Caen)[1:11]
indice_proche_Rouen = np.argsort(dist_Rouen)[-1:-11:-1]
xy = convert3D(coord)
plt.scatter(xy[:,0], -xy[:,1], color="blue")
plt.scatter(xy[indice_proche_caen,0], -xy[indice_proche_caen,1], color="gr
plt.scatter(xy[indice_proche_Rouen,0], -xy[indice_proche_Rouen,1], color="
plt.scatter(xy[ville['Caen'],0], -xy[ville['Caen'], 1], color="yellow")
plt.scatter(xy[ville["Rouen"], 0], -xy[ville['Rouen'], 1], color="yellow")
plt.show()
```

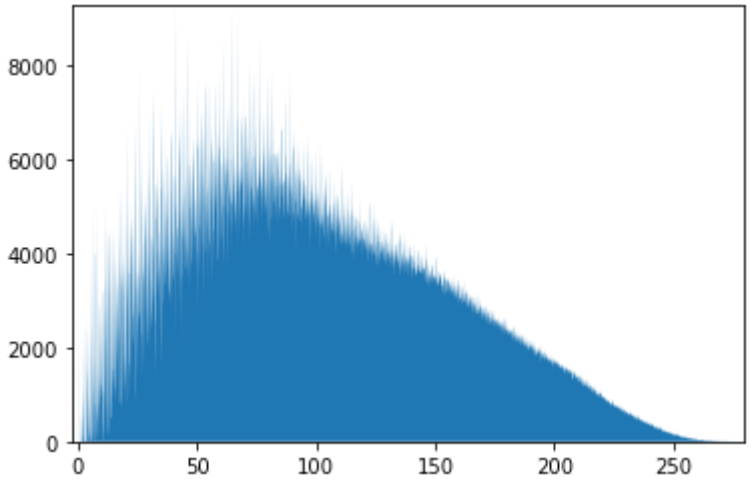


3 - Histogramme des distances des villes

Au moyen de la fonction `np.histogram` (voir le TP1) calculez et affichez l'histogramme des distances entre villes de Normandie. Le résultat devrait être proche de la figure suivante:



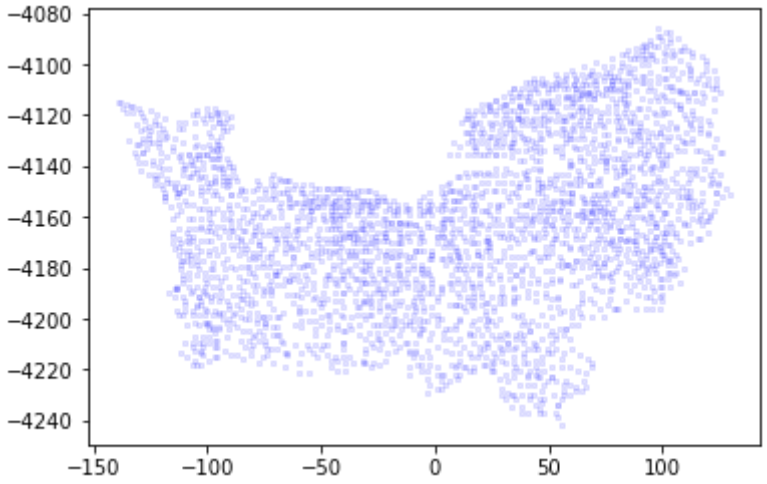
In [19]:
```
# YOUR CODE HERE
b = np.shape(d_villes)[1]
h, x = np.histogram(d_villes, bins=b)
plt.fill_between(x[:-1],0,h)
plt.axis((-1/100*(x[-1]-x[0]),x[-1]+1/100*(x[-1]-x[0]),0,np.max(h)))
plt.show()
```



4 - Le début de l'épidémie

Affichez les villes normandes à l'écran avec `matplotlib` . Ajoutez aux arguments de la fonction `plt.scatter` les valeurs `color='b',marker='s',s=5,alpha=0.1` . À votre avis que font ces arguments ?

In [20]:
```
# YOUR CODE HERE
plt.scatter(xy[:,0], -xy[:,1], color="b", marker="s", s=5, alpha=0.1)
plt.show()
```



Expliquez l'effet des arguments

Pour le début de l'épidémie nous allons considérer que chaque ville à un risque de $dfrac1500$ d'être un foyer de l'infection. Nous allons construire une fonction retournant un vecteur contenant autant d'éléments qu'il y a de villes et dont les valeurs sont soit True (si la ville est un foyer initiale de l'infection) soit False. Pour construire ce vecteur, faites les opérations suivantes :

1. Définissez une variable n contenant le nombre de villes de Normandie.
2. Tirez au hasard selon une loi uniforme un vecteur `alea` de n valeurs entre 0 et 1.
3. Testez si les valeurs du vecteur `alea` sont inférieures à $\frac{1}{500}$. Le résultat de chaque test définira le vecteur `ville_src` . Vous n'avez pas besoin de `for` pour cette question.

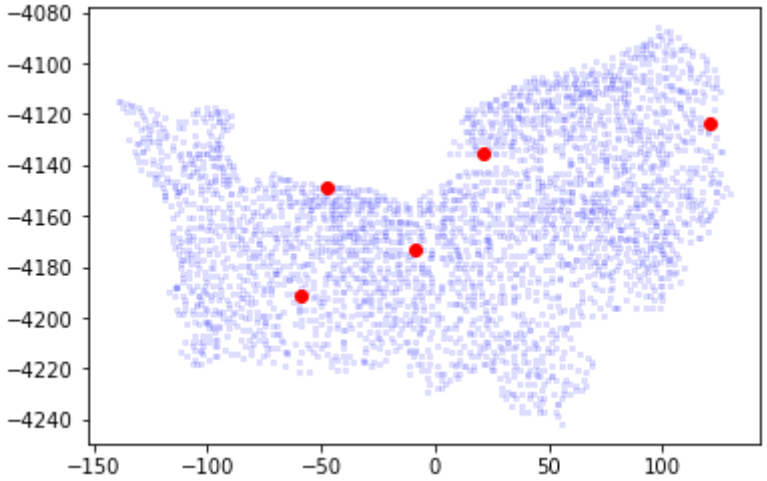
In [21]:
```
def init_foyers_epidemie():
    # YOUR CODE HERE
    n = len(ville)
    alea = np.random.uniform(0, 1,n)
    alea = alea < (500)**-1
    return alea

villes_src = init_foyers_epidemie()
```

In [22]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.

try:
    assert villes_src is not None
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('')
    print("\033[93m {}\033[00m" .format('Test 1 : La fonction init_foyers_e
    print(e)
    raise e

try:
    np.testing.assert_equal(villes_src.shape,(2946,),err_msg="\033[93m {}\(
    np.testing.assert_equal(villes_src.dtype,np.dtype('bool'),err_msg="\03
    np.testing.assert_array_less(np.sum(villes_src),20,err_msg="\033[93m {]
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

Reprenez la précédente figure. Superposez sur l'image des points rouges correspondants aux villes contaminées. Si vous relancez le code précédent, les villes en rouges devraient changer.

In [23]:
```
# YOUR CODE HERE
plt.scatter(xy[:,0], -xy[:,1], color="b", marker="s", s=5, alpha=0.1)
plt.scatter(xy[villes_src, 0], -xy[villes_src, 1], color="red")
plt.show()
```



5 - La propagation de la maladie

On va dans cette partie modéliser la propagation de la maladie. On va supposer que les villes nouvelles contaminées suivent une loi normale centrée sur les villes déjà infectées.

Définissez une variable s égale à 4. Vous pourrez changer par la suite cette valeur pour accélérer ou ralentir la propagation de la maladie.

In [24]:
```
# YOUR CODE HERE
s = 4
d_villes.shape
```

Out[24]: (2946, 2946)

Tirez selon une loi normale centrée réduite les valeurs d'une matrice nommée `nouvelle_ville_contamine_valeur` . Cette matrice aura la même taille que la matrice `d_villes` . Elle va nous permettre de choisir les villes nouvellement contaminées en fonction de leurs distances aux villes contaminées.

In [25]:
```
# YOUR CODE HERE
nouvelle_ville_contamine_valeur = np.random.standard_normal(size=np.shape(
```

In [26]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(nouvelle_ville_contamine_valeur.shape,d_villes
    np.testing.assert_array_less(np.abs(np.mean(nouvelle_ville_contamine_va
    np.testing.assert_array_less(np.abs(1-np.std(nouvelle_ville_contamine_
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

Multipliez les valeurs de la matrice `nouvelle_ville_contamine_valeur` par s pour fixer la vitesse de propagation.

In [27]:
```
# YOUR CODE HERE
nouvelle_ville_contamine_valeur = s * nouvelle_ville_contamine_valeur
```

In [28]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(nouvelle_ville_contamine_valeur.shape,d_villes
    np.testing.assert_array_less(np.abs(np.mean(nouvelle_ville_contamine_va
    np.testing.assert_array_less(np.abs(s-np.std(nouvelle_ville_contamine_v
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

Construisez une nouvelle matrice `nouvelle_ville_contamine` qui contient `True` pour les valeurs dans `nouvelle_ville_contamine_valeur` dont la valeur absolue est plus grande que la valeur dans `d_villes` et `False` pour les autres. Cette matrice permet de connaître la contamination potentielle des villes. Les villes en colonne indiquent si la ville serait contaminée dans l'hypothèse où la ville indiquée par la ligne est déjà contaminée.

In [29]:
```
# YOUR CODE HERE
nouvelle_ville_contamine = np.abs(nouvelle_ville_contamine_valeur) > d_vil
```

In [30]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(nouvelle_ville_contamine.shape,(2946, 2946),er
    np.testing.assert_equal(nouvelle_ville_contamine.dtype,np.dtype('bool'

    _tmp1,_tmp2 = np.where(nouvelle_ville_contamine)
    np.testing.assert_equal(np.all(np.abs(nouvelle_ville_contamine_valeur[_
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

Multipliez la matrice précédente par la matrice `villes_src` et sommez les valeurs suivant les colonnes. Numpy remplacera les valeurs True par 1 et False par 0 pour faire le calcul. Cela produira un vecteur qui compte pour chaque ville le nombre de villes qui sont sources de sa contamination. Si la valeur est 0, la ville n'est pas encore contaminée. Vous placerez le résultat dans la variable `villes_contaminees` .

In [31]:
```
# YOUR CODE HERE
villes_contaminees = np.sum(villes_src * nouvelle_ville_contamine, axis=1)
```

In [32]:
```
# Ce bloc permet de valider votre code. Vous ne devez pas le modifier.
try:
    np.testing.assert_equal(villes_contaminees.shape,(2946,),err_msg="\033
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

Testez si les valeurs du vecteur précédent sont supérieures strictement à 0. Si c'est le cas, c'est que la ville est contaminée.

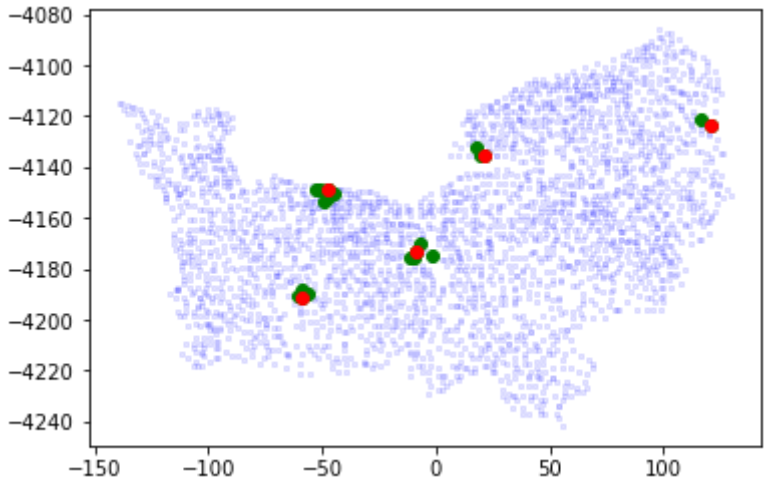
In [33]:
```
# YOUR CODE HERE
villes_contaminees = villes_contaminees > 0
```

In [34]:
```
try:
    np.testing.assert_equal(villes_contaminees.dtype,np.dtype('bool'),err_
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)
    raise e
else:
    print("\033[92m {}\033[00m" .format('Ok - Tous les tests sont validés.
```

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

Reprenez la figure précédente et affichez-y les villes nouvellement contaminées en vert. Que remarquez-vous sur les villes contaminées au début de l'épidémie ?

In [35]:
```
# YOUR CODE HERE
plt.scatter(xy[:,0], -xy[:,1], color="b", marker="s", s=5, alpha=0.1)
plt.scatter(xy[villes_contaminees, 0], -xy[villes_contaminees, 1], color="
plt.scatter(xy[villes_src, 0], -xy[villes_src, 1], color="red")
plt.show()
```

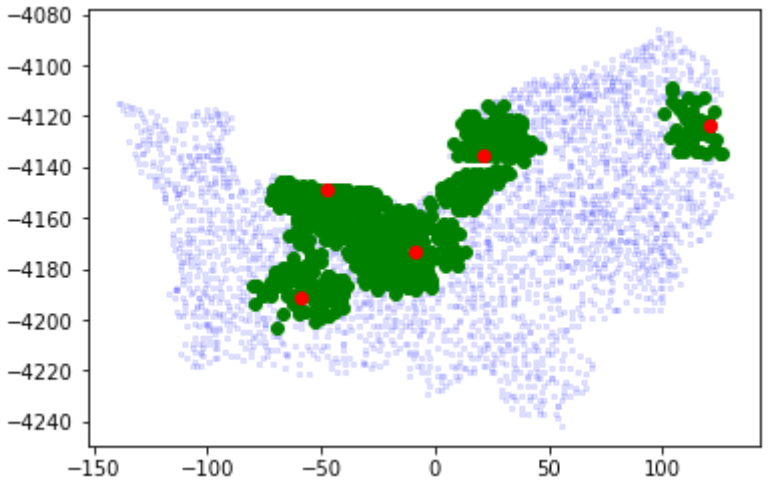


À l'aide d'une boucle `for` réitérer plusieurs fois (par exemple 5 fois) les opérations précédentes pour voir la propagation de la maladie au bout d'un certain temps. À chaque

itération, on prendra comme `villes_src` , toutes les villes précédemment contaminées.

In [36]:

```
# YOUR CODE HERE
villes_src_initiale = np.copy(villes_src)
for _ in range(5):
    nouvelle_ville_contamine_valeur = np.random.standard_normal(size=np.sha
    nouvelle_ville_contamine_valeur = s * nouvelle_ville_contamine_valeur
    nouvelle_ville_contamine = np.abs(nouvelle_ville_contamine_valeur) > d_

    villes_contaminees = np.sum(villes_src* nouvelle_ville_contamine, axis=
    villes_contaminees = villes_contaminees > 0
    villes_src += villes_contaminees

plt.scatter(xy[:,0], -xy[:,1], color="b", marker="s", s=5, alpha=0.1)
plt.scatter(xy[villes_contaminees, 0], -xy[villes_contaminees, 1], color="(
plt.scatter(xy[villes_src_initiale, 0], -xy[villes_src_initiale, 1], color=
plt.show()
```



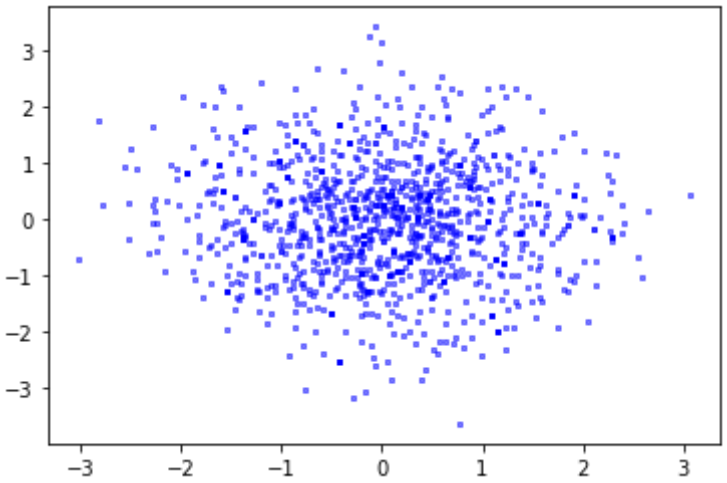
Création d'une animation (partie bonus)

Étudiez le code suivant. Que fait ce code ?

Rq : FFmpeg doit être installé sur la machine pour que le code soit fonctionel.

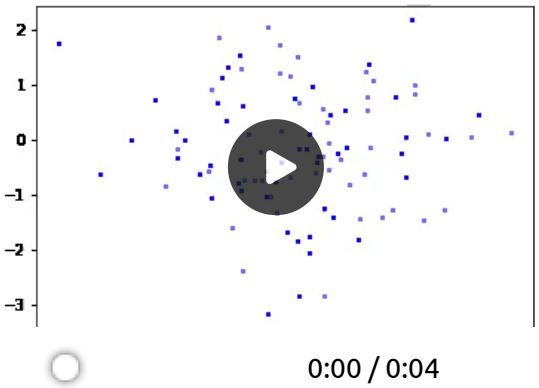
In [37]:

```
import matplotlib.animation as animation

points = np.zeros((0,2))

fig = plt.figure()
def updatefig(i):
        global points, xy
        fig.clear()
        plt.scatter(points[:,0],points[:,1],color='b',marker='s',s=5,alpha=
        x = np.random.randn(50,2)
        plt.scatter(x[:,0],x[:,1],color='b',marker='s',s=5)
        plt.draw()
        points = np.concatenate([points,x],axis=0)

anim = animation.FuncAnimation(fig, updatefig, 20)
anim.save("test.mp4", fps=5)
```



In [38]:

```
%%HTML
<video width="320" height="240" controls>
  <source src="test.mp4" type="video/mp4">
</video>
```



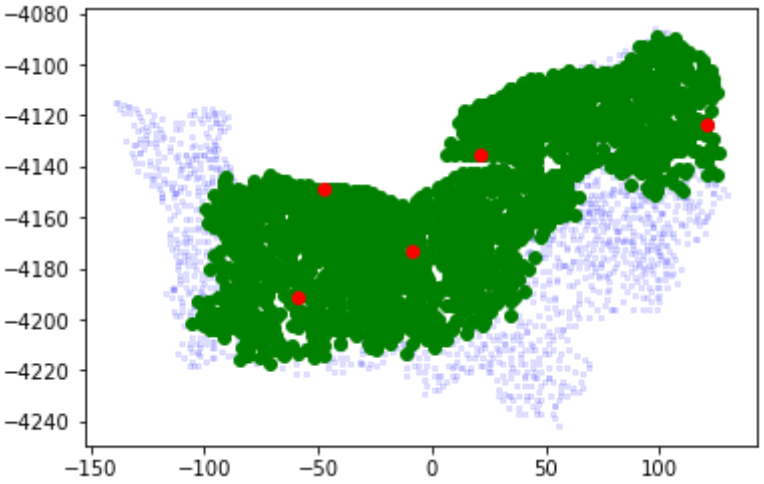
0:00 / 0:04

Ces codes permettent de generer une animation à partir des affichages des plots

Reprenez le code de la partie précédente et au lieu d'afficher le résultat à l'écran, faites une petite animation de la propagation de la maladie.

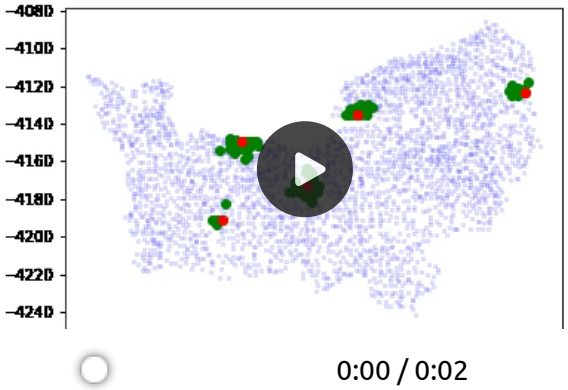
In [39]:

```
# YOUR CODE HERE

villes_orignes = np.copy(villes_src_initiale)

fig = plt.figure()
def propagation(i):
    global villes_orignes, xy
    fig.clear()
    nouvelle_ville_contamine_valeur = np.random.standard_normal(size=np.sha
    nouvelle_ville_contamine_valeur = s * nouvelle_ville_contamine_valeur
    nouvelle_ville_contamine = np.abs(nouvelle_ville_contamine_valeur) > d

    villes_contaminees = np.sum(villes_orignes* nouvelle_ville_contamine, a
    villes_contaminees = villes_contaminees > 0
    villes_orignes += villes_contaminees
    plt.scatter(xy[:,0], -xy[:,1], color="b", marker="s", s=5, alpha=0.1)
    plt.scatter(xy[villes_contaminees, 0], -xy[villes_contaminees, 1], colo
    plt.scatter(xy[villes_src_initiale, 0], -xy[villes_src_initiale, 1], co
    plt.draw()

anim2 = animation.FuncAnimation(fig, propagation, 10)
anim2.save("test2.mp4", fps=5)
```



In [40]:

```
%%HTML
<video width="320" height="240" controls>
  <source src="test2.mp4" type="video/mp4">
</video>
```



In [ ]: