Object Oriented Programming in Python
DAT171
Computer Assignment 1

Sofia Falkendal
Joel Trollheden

February 2nd 2018

Intro

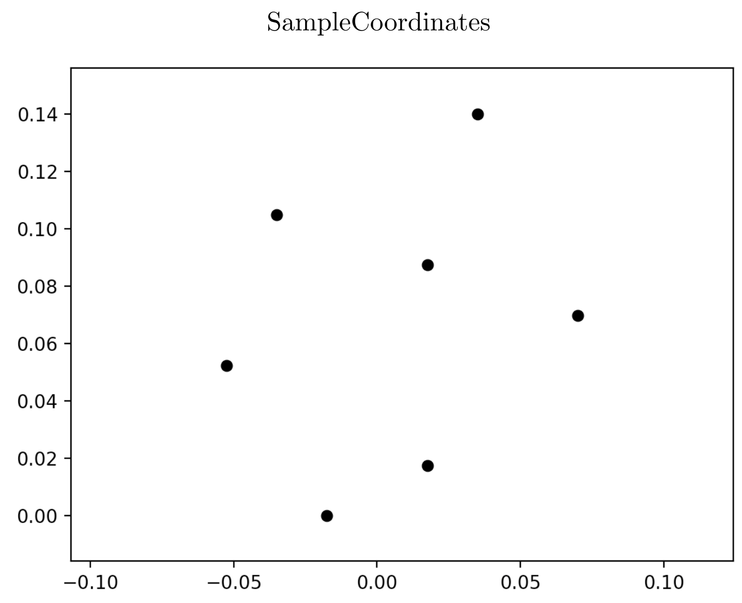
The descriptions of each task will be very short since

Task 1

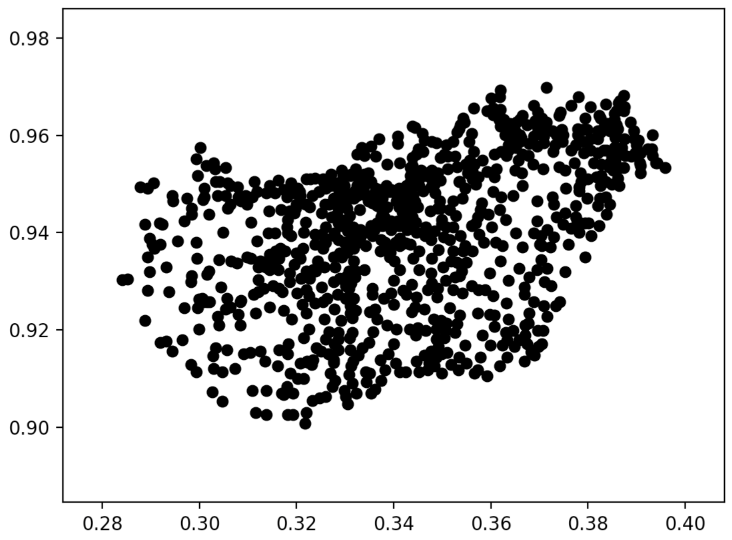
The first task was to create a list of coordinates from the text file with sample coordinates given. This was made with the function *split* and the coordinates were also converted with a Mercator projection to obtain satisfatory x and y coordinates.

Task 2

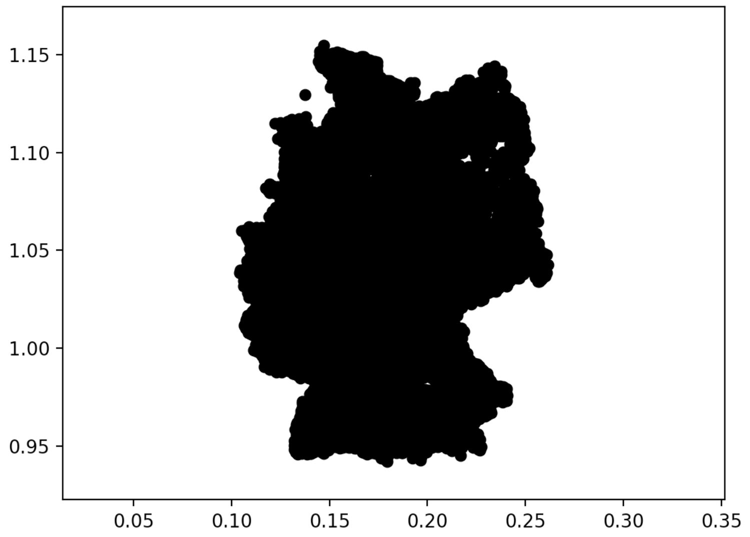
The x,y-points obtained in Task 1 were plotted using a plot function. To prevent the plot from stopping the reading of the rest of the code in the following tasks a *block* = *False* was put in the end of this task.
The plots can be seen below.



Hungary



Germany



Task 3

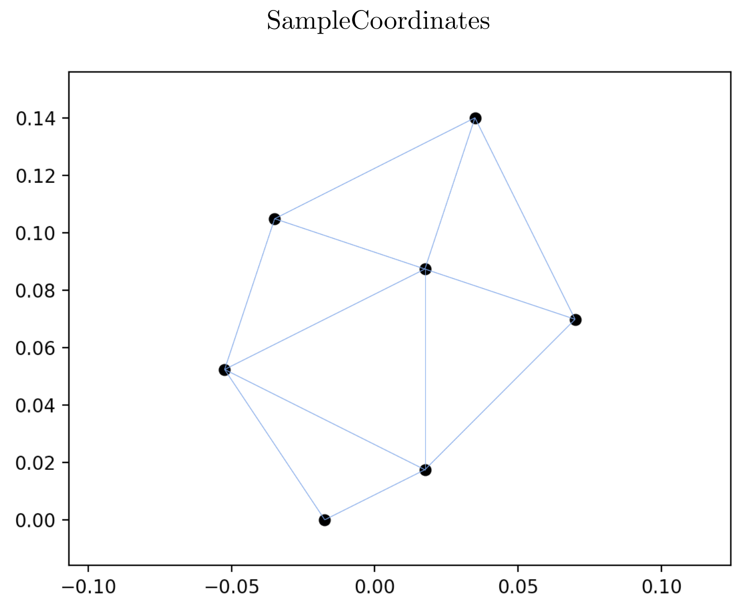
For this task the connections between the coordinates were to be obtained, excluding those that were located with the respective radius for the txt files. This was done by checking the coordinates positions against one an other and then removing those not satisfying the condition for inclusion. Two separate matrices were printed out to see the distances and connections that satisfy the radius for the qualifying coordinates.

Task 4

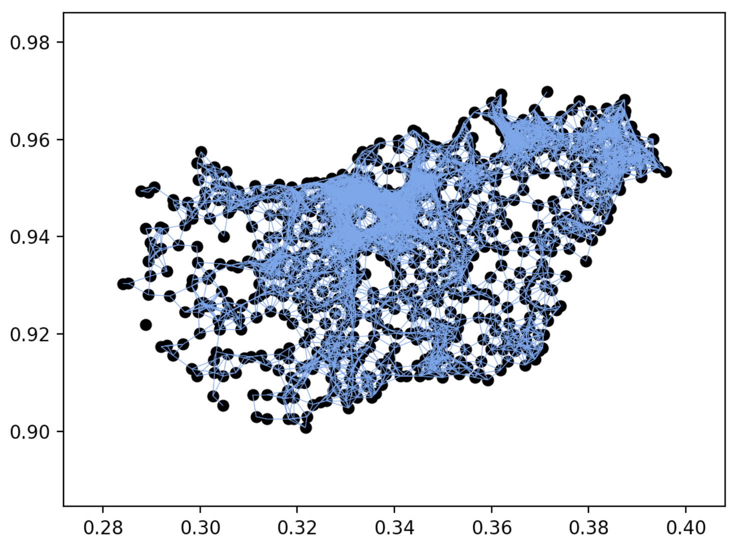
After obtaining the distances between qualifying coordinates they were to be put in a sparse graph which was to be represented in a sparse row matrix. This was done using the *csr_matrix*-function and calling on the corresponding rows and columns in the lists and matrices previously obtained. This is to make it easier to process and visualise the data. The matrix that comes out is a NxN matrix (Every point (N points) has a specific distance to every other point (N points) and thus the NxN matrix is born.

Task 5

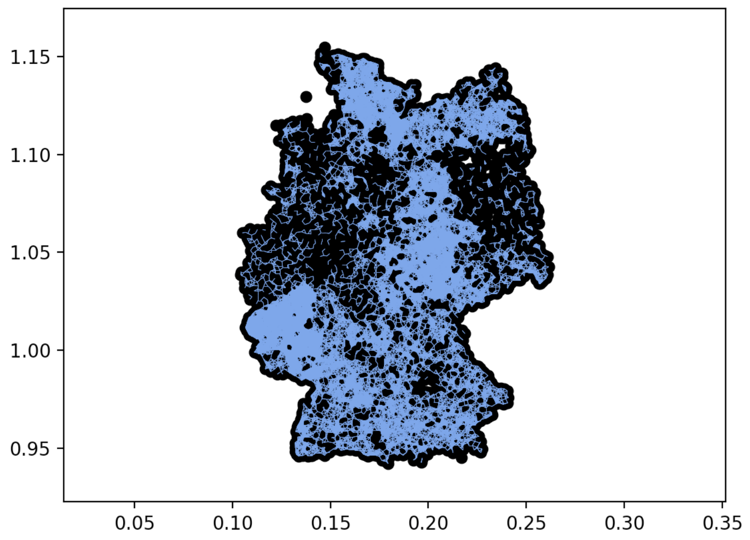
What had now been found in tasks 3 and 4 was to be represented graphically by plotting lines between the points that satisfied the conditions of maximum distance. *Line_Collection* was used in order to make the run of the code faster. The plots can be seen below.



Hungary



Germany



Task 6

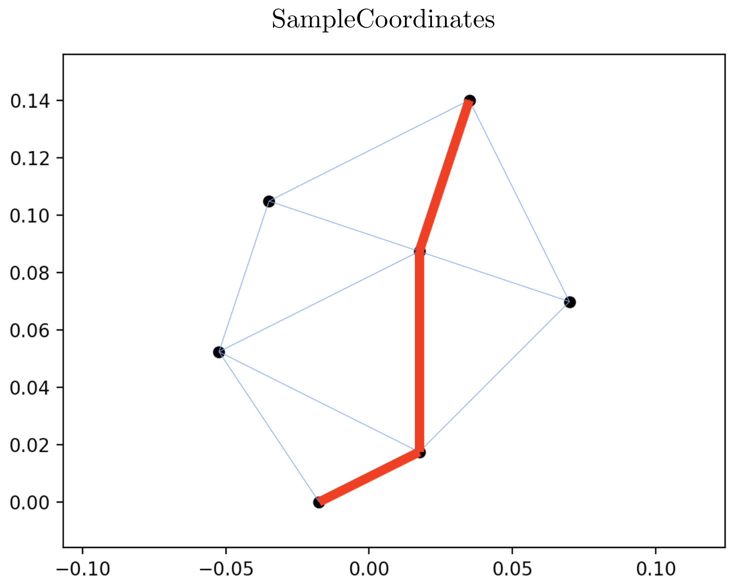
Once the different paths had been found, the shortest path was to be shown. The *dijkstra*-function calculates the distances from each point to every other point and also gives the predecessor node. The paths can be found under task 9 due to a printscreen containing both time and optimal path.

Task 7

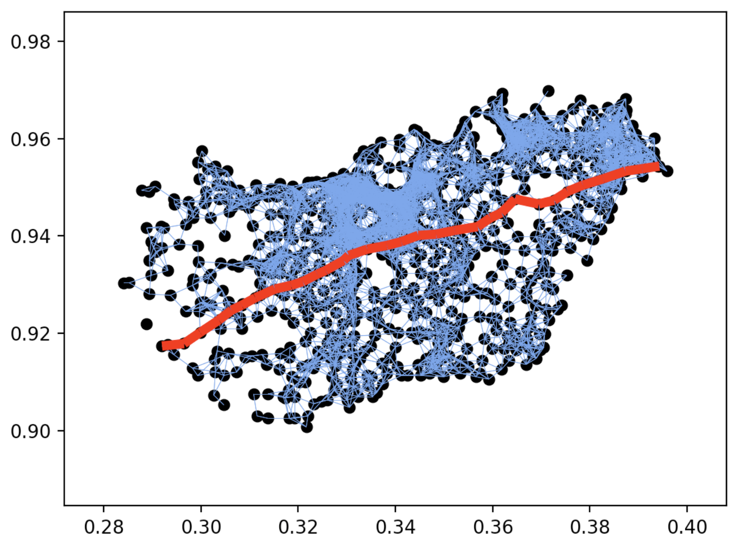
In task 7 the function computepath the predecessor data is used to create the indices for the shortest path from the start node to the end node.

Task 8

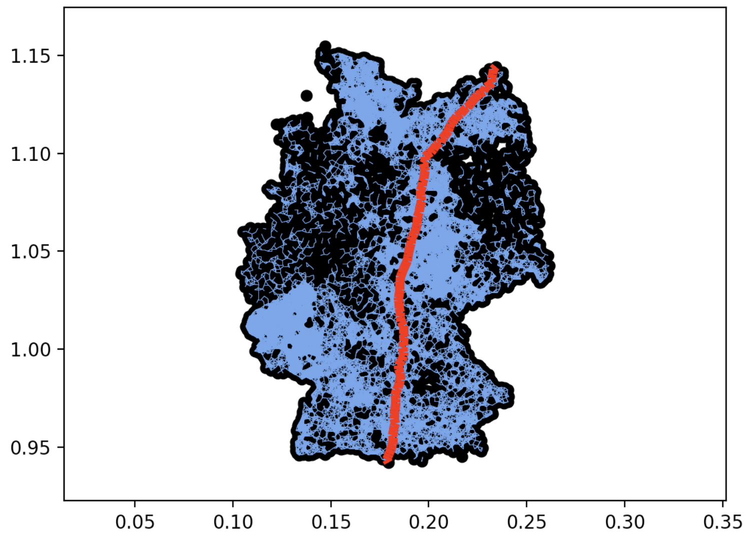
A new part is added to the plot function so that the shortest path is ploted out in the graph over the coordinates. The plots are shown below:



Hungary



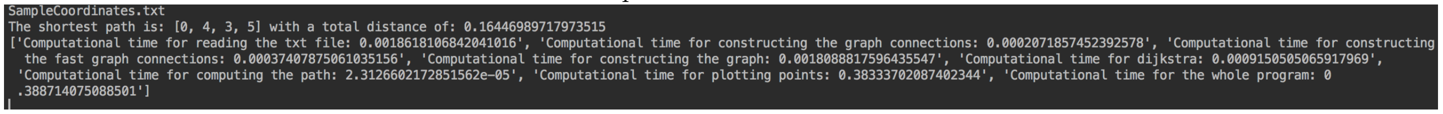
Germany



Task 9

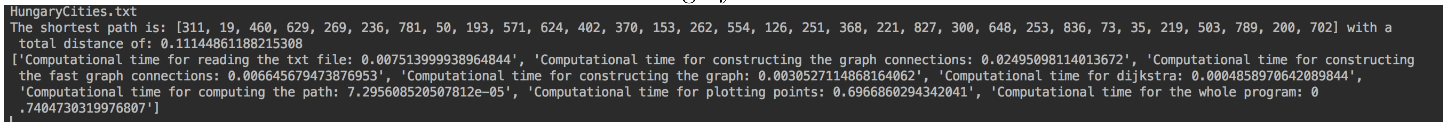
The computational times (and most optimal path) for the different coordinates (Germany, Hungary and SampleCoordinates) are the following:

SampleCoordinates



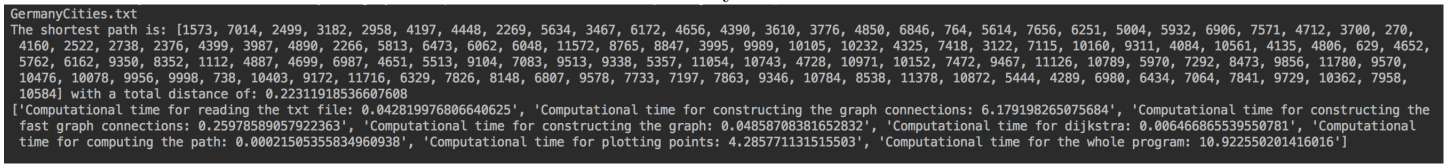
```
SampleCoordinates.txt
The shortest path is: [0, 4, 3, 5] with a total distance of: 0.16446989717973515
['Computational time for reading the txt file: 0.0018618106842041016', 'Computational time for constructing the graph connections: 0.0002071857452392578', 'Computational time for constructing
  the fast graph connections: 0.00037407875061035156', 'Computational time for constructing the graph: 0.0018088817596435547', 'Computational time for dijkstra: 0.0009150505065917969',
 'Computational time for computing the path: 2.3126602172851562e-05', 'Computational time for plotting points: 0.38333702087402344', 'Computational time for the whole program: 0
 .388714075088501']
```

Hungary



```
HungaryCities.txt
The shortest path is: [311, 19, 460, 629, 269, 236, 781, 50, 193, 571, 624, 402, 370, 153, 262, 554, 126, 251, 368, 221, 827, 300, 648, 253, 836, 73, 35, 219, 503, 789, 200, 702] with a
 total distance of: 0.11144861188215308
['Computational time for reading the txt file: 0.007513999938964844', 'Computational time for constructing the graph connections: 0.02495098114013672', 'Computational time for constructing
 the fast graph connections: 0.006645679473876953', 'Computational time for constructing the graph: 0.0030527114868164062', 'Computational time for dijkstra: 0.0004858970642089844',
 'Computational time for computing the path: 7.295608520507812e-05', 'Computational time for plotting points: 0.6966860294342041', 'Computational time for the whole program: 0
 .7404730319976807']
```

Germany



```
GermanyCities.txt
The shortest path is: [1573, 7014, 2499, 3182, 2958, 4197, 4448, 2269, 5634, 3467, 6172, 4656, 4390, 3610, 3776, 4850, 6846, 764, 5614, 7656, 6251, 5004, 5932, 6906, 7571, 4712, 3700, 270,
 4160, 2522, 2738, 2376, 4399, 3987, 4890, 2266, 5813, 6473, 6062, 6048, 11572, 8765, 8847, 3995, 9989, 10105, 10232, 4325, 7418, 3122, 7115, 10160, 9311, 4084, 10561, 4135, 4806, 629, 4652,
 5762, 6162, 9350, 8352, 1112, 4887, 4699, 6987, 4651, 5513, 9104, 7083, 9513, 9338, 5357, 11054, 10743, 4728, 10971, 10152, 7472, 9467, 11126, 10789, 5970, 7292, 8473, 9856, 11780, 9570,
 10476, 10078, 9956, 9998, 738, 10403, 9172, 11716, 6329, 7826, 8148, 6807, 9578, 7733, 7197, 7863, 9346, 10784, 8538, 11378, 10872, 5444, 4289, 6980, 6434, 7064, 7841, 9729, 10362, 7958,
 10584] with a total distance of: 0.22311918536607608
['Computational time for reading the txt file: 0.042819976806640625', 'Computational time for constructing the graph connections: 6.179198265075684', 'Computational time for constructing the
 fast graph connections: 0.2597858905792236З', 'Computational time for constructing the graph: 0.04858708381652832', 'Computational time for dijkstra: 0.006466865539550781', 'Computational
 time for computing the path: 0.00021505355834960938', 'Computational time for plotting points: 4.285771131515503', 'Computational time for the whole program: 10.922550201416016']
```

Task 10

In this part a function is created to construct the graph connections with the cKDTree functions to minimize the time that the function takes compared to the function constructed in task 3.