IMPERIAL COLLEGE LONDON, DEPARTMENT OF MATHEMATICS

Scientific Computing (M3SC) Project 1

Omar Haque

February 22, 2017

1 MAIN SOLUTION

The code in *solution.py* contains the main program which carries out the process outlined by the question, i.e modelling the process of the cars moving across the city of Rome using the rules described. I have added scripts *to be added* and *to be added* to help answer the related questions at the end of the project.

Below are the imports and functions used by the main program.

```
# Imports
import Dijkstra as dijk
import misc
import numpy as np
import csv

# This import is needed for the last question
from solution_accident_occurs import max_index_tracker_no30


# ------------------------------------------------
# ------------    FUNCTIONS USED    --------------
# ------------------------------------------------

def next_node(path):
    """ Returns the next index (after the node itself) in the path.
```

```
        If the path contains only one node, returns the node itself.
    """
    if len(path) == 1:
        return path[0]
    else:
        return path[1]


def update_weight_matrix(epsilon, c, original_weight_matrix, noNodes=58):
    """
    This function updates the weight matrix according to step 5 of the
    Project. Note the added fix - the weight matrix is not changed if
    the original entry was 0.



    :param epsilon: given in question
    :param c: the vector containing number of cars at each node
    :param original_weight_matrix: the weight matrix given by RomeEdges
    :param noNodes: number of nodes in the system
    :return: the updated weight matrix
    """
    new_weight_matrix = np.zeros((noNodes, noNodes))
    for i in range(noNodes):
        for j in range(noNodes):
            if original_weight_matrix[i, j] != float(0):
                new_weight_matrix[i, j] = original_weight_matrix[i, j] + \
                                          (epsilon * (float(c[i]) +
                                                      float(c[j]))) / float(2)
    return new_weight_matrix


def extract_data():
    """
    This function opens the RomeVertices and RomeEdges files, and creates
    global variables RomeX, RomeY, RomeA, RomeB and RomeV. These are variables
    used to create the original weight matrix.

    """
    global RomeX, RomeY, RomeA, RomeB, RomeV
    RomeX = np.empty(0, dtype=float)
    RomeY = np.empty(0, dtype=float)
    with open('./data/RomeVertices', 'r') as file:
        AAA = csv.reader(file)
```

```
61          for row in AAA:
62              RomeX = np.concatenate((RomeX, [float(row[1])]))
63              RomeY = np.concatenate((RomeY, [float(row[2])]))
64      file.close()
65      RomeA = np.empty(0, dtype=int)
66      RomeB = np.empty(0, dtype=int)
67      RomeV = np.empty(0, dtype=float)
68      with open('./data/RomeEdges2', 'r') as file:
69          AAA = csv.reader(file)
70          for row in AAA:
71              RomeA = np.concatenate((RomeA, [int(row[0])]))
72              RomeB = np.concatenate((RomeB, [int(row[1])]))
73              RomeV = np.concatenate((RomeV, [float(row[2])]))
74      file.close()
```

My solution for moving the cars across the graph are as follows.
For each iteration (minute)