IMPERIAL COLLEGE LONDON, DEPARTMENT OF MATHEMATICS

Scientific Computing (M3SC)

Ronak Shah

February 23, 2017

PROJECT 1: TRAFFIC FLOW THROUGH A SIMPLIFIED STREET NETWORK

1 CODE EXPLANATION

Initially, in the top three lines, I import *NumPy*, *CSV* (Comma Separated Values) and *math* packages.

```
import numpy as np
import csv
import math
```

I then go on to import the *Dijkstra* algorithm and *calcWei* code from tutorial 3, so I can use it freely in my code.

```
from Dijkstra import Dijkst, calcWei
```

I then set up the global variables in such a way that one can just change the *START* and *END* nodes to execute alternate simulations. I go on to generate a list of 58 nodes (corresponding to the 58 vertices in the *RomeEdges* and then remove node 52 as I will treat this as an edge case due to the nature of the problem when only one car remains at node 52.

```
XI = 0.01
NUMNODES = 58
START = 12 # St. Peter's Square
END = 51 # Coliseum
```

```
RANGEEXEND = range(NUMNODES)
RANGEEXEND.remove(END) #Range of nodes, without the END node
```

I then define the function *printCars*, which takes *cars* as its argument. *printCars* is just there to output the simulation with appropriate node numbering and format.

updateWij takes arguments *RomeA*, *RomeB*, *wij*, *wijZeroth* and *cars*. This function iterates over the weights to update them as per the following formula:

$$w_{ij} = w_{ij}^{(0)} + \xi \frac{c_i + c_j}{2}.$$

```
def updateWij(RomeA, RomeB, wij, wijZeroth, cars):
    #Updates the weights according to the given formula
    for n in range(len(RomeA)):
        i = RomeA[n]-1
        j = RomeB[n]-1
        wij[i,j] = wijZeroth[i,j]+(XI * ((cars[i]+cars[j])/2))
```

In *RomeSimulate* we are finding the next optimal node to go to according to Dijkstra's algorithm. This includes 70% of cars leaving each node and moving onto the next node. As well as 60% of the cars at node 52 remaining due to traffic. Below, I am also using an if loop inside a for loop to set the used edges to zero (so later we can determine the number of unused edges for Question 3).

```
def RomeSimulate(RomeA,RomeB,wij,wijZeroth,cars,nodeLoad,
    nextNode,unusedEdges):
    #find optimal route for each node to node END
    for i in RANGEEXEND:
        #The 1st elem of the returned array is next node to travel to
        nextNode[i] = int(Dijkst(i,END,wij)[1])
        #setting unused edges to zero in unusedEdges
        if cars[i] != 0:
            unusedEdges[i, nextNode[i]] = 0

    #Node END. 60% of cars stay.
    carsAtEND = cars[END]
    nodeLoad[END] = max(nodeLoad[END], carsAtEND)
    cars[END] = round(0.6 * carsAtEND)

    #Move the cars at all other nodes except END
    copyCars = np.copy(cars)
    for i in RANGEEXEND:
        carsAtNode = copyCars[i]
        nodeLoad[i] = max(nodeLoad[i], carsAtNode)
```

```
        oldCars = cars[i]
        carsLeft = round(oldCars - 0.7 * carsAtNode)
        cars[i] = carsLeft
        carsMoved = oldCars - carsLeft
        cars[int(nextNode[i])] += carsMoved

    #update wij
    updateWij(RomeA, RomeB, wij, wijZeroth, cars)
    return
```

I then proceed with the following executable code. Initially I set up *RomeX* & *RomeY* to be the vertices from the *RomeVertices* file and *RomeA*, *RomeB* & *RomeV* to be the edges and weights from the *RomeEdges* file.

```
if __name__ == "__main__":
    #Get the x and y values of Romes' vertices
    RomeX = np.empty(0,dtype=float)
    RomeY = np.empty(0,dtype=float)
    with open('RomeVertices','r') as file:
        AAA = csv.reader(file)
        for row in AAA:
            RomeX = np.concatenate((RomeX,[float(row[1])]))
            RomeY = np.concatenate((RomeY,[float(row[2])]))
    file.close()

    #Get the edge values and initial weight values of Romes' edges
    unusedEdges = np.zeros(shape=(NUMNODES, NUMNODES))
    RomeA = np.empty(0,dtype=int)
    RomeB = np.empty(0,dtype=int)
    RomeV = np.empty(0,dtype=float)
    with open('RomeEdges','r') as file:
        AAA = csv.reader(file)
        for row in AAA:
            RomeA = np.concatenate((RomeA,[int(row[0])]))
            RomeB = np.concatenate((RomeB,[int(row[1])]))
            usedEdges[int(row[0])-1, int(row[1])-1] = 1
            RomeV = np.concatenate((RomeV,[float(row[2])]))
    file.close()
```

I then initialise *cars*, *nodeLoad* and *nextNode*. Then, calculate the initial weights of the edges:

```
        #Initialize arrays
    cars = np.zeros(shape=NUMNODES)
    nodeLoad = np.zeros(shape=NUMNODES)
    nextNode = np.zeros(shape=NUMNODES)
```

```

    #Calculate initial weights of edges
    wij = calcWei(RomeX, RomeY, RomeA, RomeB, RomeV)
    wijZeroth = np.copy(wij)
```

And finally, the code to print out:

- the 200 iterations with numbering
- list of the nodes with their maximum loads *(Question 1)*
- the top five most congested nodes *(Question 2)*
- the number of unused edges *(Question 3)*

```
#print "Simulating Rome\nInitial state:"
    printCars(cars)
    for i in range(200):
        if i < 180:
            cars[START] += 20
        print "############################################"
        print "Simulating iteration: ", i+1
        RomeSimulate(RomeA,RomeB,wij,wijZeroth,cars,nodeLoad,
        nextNode,unusedEdges)
        printCars(cars)

    #calculating unused edges using nested for loops
    notUsedEdges = []
    for i in range(NUMNODES):
        for j in range(NUMNODES):
            if unusedEdges[i,j] == 1:
                notUsedEdges.append([i+1,j+1])

    #We can use same func for printing cars to print the load
    #for each node
    print "Maximum Node Load:"
    printCars(nodeLoad)

    #Calculating the top 5 most congested nodes
    MostCongestedNodes = nodeLoad.argsort()[-5:][::-1]+1
    print 'Top Five Most Congested Edges:', MostCongestedNodes

    #print the unused edges and the number of them
    print 'List of Unused Edges:', notUsedEdges
    print 'Number of Unused Edges:', np.sum(unusedEdges)
```

2 QUESTIONS

QUESTION 1

The maximum load for each node after 200 iterations is as follows:

1: 4.0	2: 6.0	3: 0.0	4: 7.0
5: 0.0	6: 11.0	7: 9.0	8: 0.0
9: 16.0	10: 12.0	11: 0.0	12: 8.0
13: 28.0	14: 0.0	15: 28.0	16: 23.0
17: 7.0	18: 28.0	19: 9.0	20: 29.0
21: 39.0	22: 16.0	23: 8.0	24: 24.0
25: 40.0	26: 17.0	27: 7.0	28: 12.0
29: 12.0	30: 27.0	31: 11.0	32: 16.0
33: 17.0	34: 15.0	35: 20.0	36: 9.0
37: 6.0	38: 15.0	39: 15.0	40: 30.0
41: 34.0	42: 10.0	43: 27.0	44: 30.0
45: 4.0	46: 0.0	47: 0.0	48: 12.0
49: 0.0	50: 23.0	51: 19.0	**52: 62.0**
53: 16.0	54: 14.0	55: 12.0	56: 13.0
57: 11.0	58: 11.0		

QUESTION 2

To find the five most congested nodes we employ the use of the following code which sorts the loads from the above array in descending order and takes the top five.

```
#Calculating the top 5 most congested nodes
MostCongestedNodes = nodeLoad.argsort()[-5:][::-1]+1
print 'Top Five Most Congested Edges:', MostCongestedNodes
```

Output:

```
Top Five Most Congested Edges: [52 25 21 41 44]
```

Therefore the five most congested nodes are:

1. Node: **52** *(Maximum load = 62)*
2. Node: **25** *(Maximum load = 40)*
3. Node: **21** *(Maximum load = 39)*
4. Node: **41** *(Maximum load = 34)*
5. Node: **44** (& 40) *(Maximum load = 30)*

Question 3

To find the edges which are not utilised at all, we employ the following lines of code to print out the list of unused edges and the number of unused edges.

```
    #print the unused edges and the number of them
    print 'List of Unused Edges:', notUsedEdges
    print 'Number of Unused Edges:', np.sum(unusedEdges)
```

The output is:

```
List of Unused Edges: [[1, 2], [1, 7], [2, 3], [3, 2], [3, 5],
[4, 1], [4, 7], [5, 8], [7, 1], [7, 6], [8, 9], [8, 11],
[9, 8], [10, 9], [11, 14], [11, 16], [12, 4], [12, 17],
[14, 15], [14, 18], [15, 13], [16, 11], [17, 19], [18, 15],
[19, 10], [21, 18], [23, 12], [27, 23], [28, 17], [28, 29],
[29, 19], [29, 34], [30, 21], [31, 27], [31, 36], [32, 22],
[32, 26], [33, 22], [33, 32], [34, 33], [35, 30], [36, 28],
[37, 24], [38, 35], [39, 38], [41, 29], [41, 39], [41, 43],
[42, 31], [42, 44], [43, 30], [43, 49], [44, 36], [44, 41],
[44, 42], [45, 30], [45, 46], [46, 37], [46, 45], [47, 48],
[48, 45], [48, 46], [49, 43], [49, 47], [52, 42], [52, 44],
[52, 58], [54, 49], [55, 56], [56, 54], [57, 55], [58, 57]]
```

```
Number of Unused Edges: 72.0
```

The reason these edges are not used is because when Dijkstra's algorithm is running it is continually updating and finding the shortest path to the sink node (which, in our case, is node 52). During these iterations Dijkstra's algorithm is receving information about the weights of each edges which is connected to the node it is currently at. Clearly the weights of the edges above were too high and thus deemed inefficient so Dijkstra's algoritm chose an alternate, more efficient, edge to travel along from that node to the next.

Question 4

Setting $\xi = 0$ gives us:

$$w_{ij} = w_{ij}^{(0)}.$$

Hence, the weights no longer update with each iteration. So with $\xi = 0$ we get the new maximum loads for each node as follows:

1: 0.0	2: 0.0	3: 0.0	4: 0.0
5: 0.0	6: 0.0	7: 0.0	8: 0.0
9: 0.0	10: 0.0	11: 0.0	12: 0.0
13: 28.0	14: 0.0	15: 28.0	16: 0.0
17: 0.0	18: 28.0	19: 0.0	20: 0.0
21: 0.0	22: 0.0	23: 0.0	24: 0.0
25: 28.0	26: 0.0	27: 0.0	28: 0.0
29: 0.0	30: 0.0	31: 0.0	32: 0.0
33: 0.0	34: 0.0	35: 0.0	36: 0.0
37: 0.0	38: 0.0	39: 0.0	40: 28.0
41: 0.0	42: 0.0	43: 0.0	44: 0.0
45: 0.0	46: 0.0	47: 0.0	48: 0.0
49: 0.0	50: 28.0	51: 28.0	**52: 49.0**
53: 28.0	54: 28.0	55: 28.0	56: 28.0
57: 28.0	58: 28.0		

So we observe that since the weights are not updating, Dijkstra's algorithm chooses one path and sticks to it. Namely:

13 - 15 - 18 - 25 - 40 - 50 - 51 - 53 - 54 - 55 - 56 - 57 - 58 - 52.

In addition, with $\xi = 0$, we observe:

```
Top Five Most Congested Edges: [52 58 50 57 18]
```

And unsurprisingly, we observe:

```
Number of Unused Edges: 141.0
```

Question 5

To implement the accident at node 30 I introduce the function: *blockNode30*, which takes *wijZeroth* as its argument. To ensure Dijkstra's algorithm never chooses node 30, I have made the weights of all edges going to node 30 to be of very high weight so Dijkstra will not choose node 30 in the algorithm.

```
def blockNode30(wijZeroth):
    wijZeroth[29, 25] = 9999
    wijZeroth[29, 34] = 9999
    wijZeroth[29, 42] = 9999
    wijZeroth[29, 44] = 9999
    wijZeroth[29, 20] = 9999

blockNode30(wijZeroth)
```

With the accident at node 30, the new maximum loads for each node is as follows:

1: 6.0	2: 9.0	3: 0.0	4: 11.0
5: 0.0	6: 14.0	7: 13.0	8: 0.0
9: 20.0	10: 15.0	11: 0.0	12: 10.0
13: 28.0	14: 0.0	15: 28.0	16: 23.0
17: 7.0	18: 28.0	19: 11.0	20: 26.0
21: 29.0	22: 16.0	23: 9.0	24: 19.0
25: 35.0	26: 19.0	27: 8.0	28: 12.0
29: 14.0	30: 0.0	31: 11.0	32: 22.0
33: 17.0	34: 16.0	35: 16.0	36: 10.0
37: 0.0	38: 14.0	39: 12.0	40: 26.0
41: 21.0	42: 10.0	43: 16.0	44: 23.0
45: 0.0	46: 0.0	47: 0.0	48: 15.0
49: 0.0	50: 20.0	51: 19.0	**52: 58.0**
53: 16.0	54: 14.0	55: 11.0	56: 12.0
57: 11.0	58: 10.0		

As expected the maximum load for node 30 is now 0. With the accident, the most congested nodes are:

```
Top Five Most Congested Edges: [52 25 21 13 18]
```

Therefore the most congested nodes are:

1. Node: **52** *(Maximum load = 58)*
2. Node: **25** *(Maximum load = 35)*
3. Node: **21** *(Maximum load = 29)*
4. Node: **13** (& 18 and 19) *(Maximum load = 28)*

The nodes which decreased and increased the most in their maximum loads are:

Nodes	Maximum Loads		Load Change
	Without Accident	With Accident	
41	34	21	-13
43	27	16	-11
21	39	29	-10
44	30	23	-7
32	16	22	+6
4	7	11	+4
7	9	13	+4
9	16	20	+4

In addition, with the accident at node 30, we observe:

```
Number of Unused Edges: 83.0
```