Introduction to Traffic Flow Modeling and Intelligent Transport Systems
Macroscopic Fundamental Diagram

Tristan Ford

December 18, 2021

Abstract

In this lab we will do an analysis of traffic data in urban networks. The provided data is extracted from a micro-simulation environment that replicates the real measurements of inductive loop detectors in cities. We will be able to estimate some common traffic characteristics (flow, density, occupancy, speed) and plot relations between them. We will estimate the Macroscopic Fundamental Diagram of flow vs. density (or production vs. accumulation) for different spatial scales and identify how this scaling changes the results.

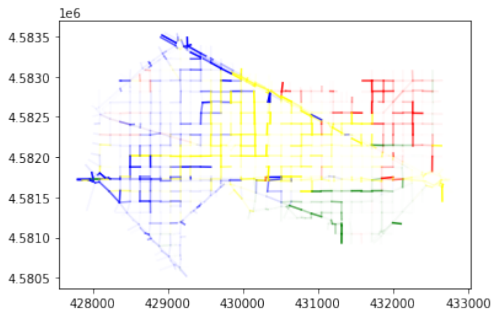
I have utilized Python to perform the analysis and generate all plots and figures. Code is included as an appendix for reference.

1 Step 1

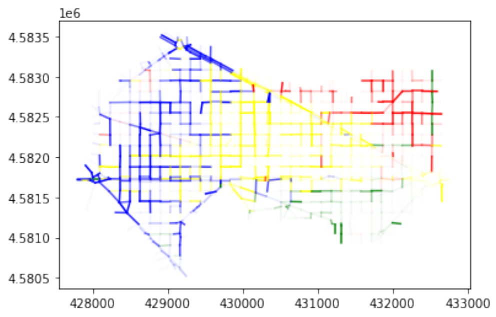
Create three snapshots of the network presenting the level of congestion of each link at time 60min, 90min and 120min of the simulation. Use the provided gplotdc.m file for plotting the network in a gray-scale format in which the color denotes the level of congestion for each link. Use the occupancy measurements [0, 100] to specify how dark should be the color of each link (i.e., links with occupancy 0 and 100 are shown with white and black color respectively).

First, we generate lists from the provided data. Then, using the occupancy measure, plot each link with a corresponding transparency and color for its region.

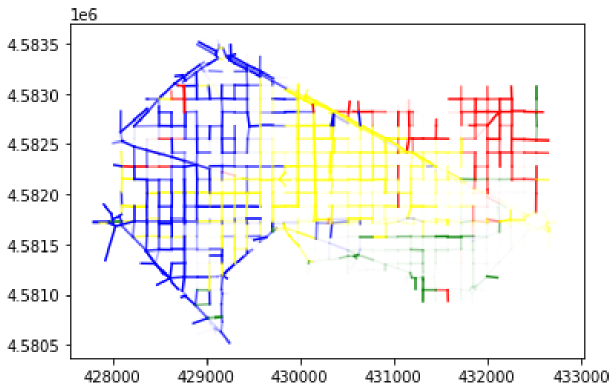
- Region 1;
- Region 2;
- Region 3; and
- Region 4.



(a) 60 minutes



(b) 90 minutes



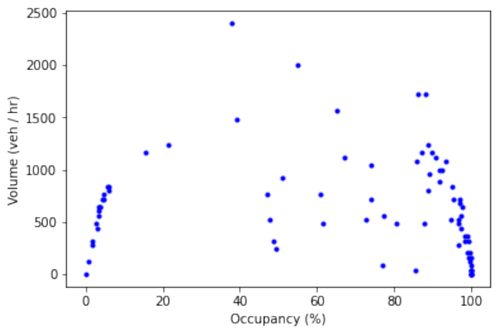
(c) 120 minutes

Figure 1: Evolution of occupancy within the network.

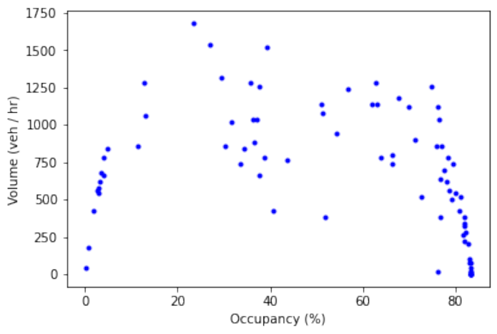
2 Step 2

Traffic volume V in traffic engineering is typically expressed in terms of vehicles per hour. Convert all the flow measurements in the data file to an equivalent expressed in vehicles per hour. Pick one link of the network and produce a scatter plot of volume vs. occupancy for the 90-seconds intervals. Do the same scatter plot for the average volume vs. average occupancy of two links. Finally, produce the scatter plot of average volume vs. average occupancy for all network links. Do the same plot for each of the 4 regions of the network. Is there any pattern to the relationship between average volume and average occupancy? Comment on the partitioning of the network.

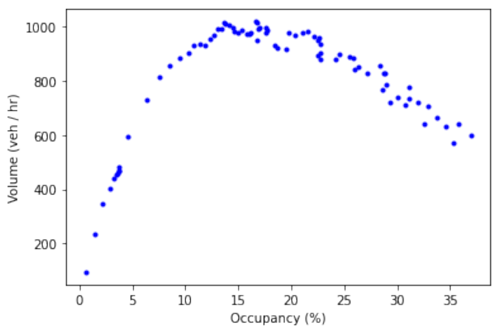
First, we generate volumes from the provided data. Then, we plot the first three figures. To obtain the last figure separating the various regions a few more lines are required.



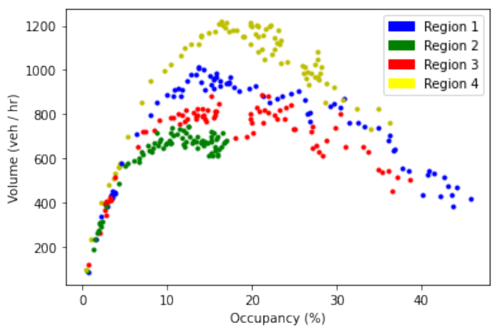
(a) Random single link



(b) Average of two random links



(c) Average of all links



(d) Average of all links

Figure 2: Volume versus occupancy for various combinations of network links.

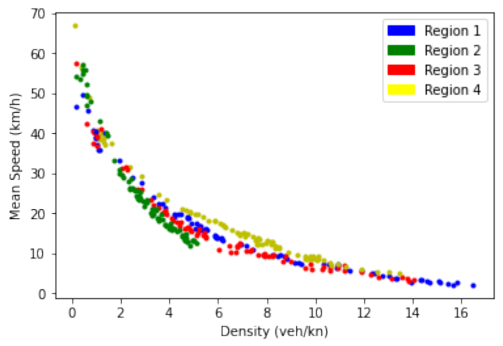
The pattern we observe is the relationship between volume (or flow) and occupancy. This is one representation of the fundamental diagram for this network - the macroscopic fundamental diagram (MFD).

Looking at figure 2d, we can observe that region 4 has the highest volume which may indicate this is the most utilized region in the network. Regions 1 and 3 appear to be well utilized partitions of the network and they both reach higher levels of occupancy which may indicate these regions experience high levels of congestion. This makes sense as their MFD's follow similar trends. Region 2 appears to be the least utilized in the network and does not surpass 20 % occupancy on average. With less utilization we can expect the volumes in this region to be lower which is easily seen in the figure.

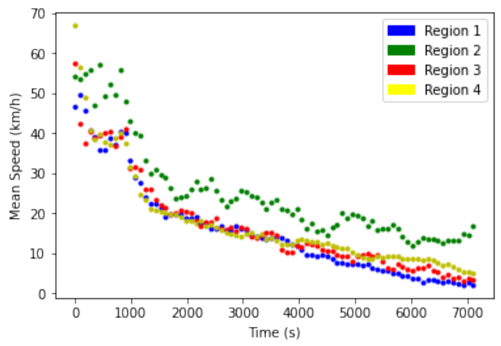
3 Step 3

Produce a scatter plot for each region with mean speed as a function of average density of the region. Is there any pattern to the relationship? Plot also the time-series of the region mean speed for the two hours of simulation. Does the range of values of the mean speed make sense for an urban network? What do the time series reveal for the congestion of the network?

For each region and time step, we calculate the density and volume. Using these we can plot the mean speed against time and density.



(a) Mean speed versus density



(b) Mean speed evolution over time

Figure 3: Relationship between mean speed, time and density across the four regions in the network.

Mean speed versus density appears to follow a $1/x$ relationship as seen in figure 3a. This makes sense as more vehicles populate the road, we can expect the average speeds to decrease.

Looking at figure 3b, we can see that over time mean speeds throughout the network decrease. Assuming the simulation occurs during a peak period this would be consistent with traffic building up as people commute and go about their daily activities. The speeds are consistent with what we might expect in an urban setting. Speeds max out at around 70 km/h and settle down to 20-30 km/h during congested periods.

Regions 1, 3 and 4 follow a very similar mean speed vs. time trend which indicates the level of congestion in these networks are linked closely. We can see that region 2 has slightly higher mean speeds which is consistent with our previous hypothesis that this region is less congested than the rest of the network. Interestingly, there is some cyclical variation in mean speed for region 2 over time. This may be a result of the region begin less frequented which could produce more scatter in the data.

4 Step 4

Produce a scatter plot of production vs. accumulation for one random link. Do the same plot for the summation of the variables for two links. Finally, produce scatter plots for all the regions displaying the total production of the region vs. the accumulation (total number of vehicles in the region). What is the difference between these MFDs and the ones generated in Step 2?

To simplify the analysis, I have constructed only a plot for the production vs. accumulation for all four regions. To do so, I followed a similar process to the one in Step 2 to calculate values for accumulation and production across the different regions. I then fit a 3rd degree polynomial to each data set.

Would we have plotted the single, double and total link relationships I would expect a similar pattern to emerge. Single and double link plots would have significant scatter in the center of the MFD due to high variability of traffic on a single link throughout the day. The total link plot would follow the characteristic MFD curve which averages across the various regions.

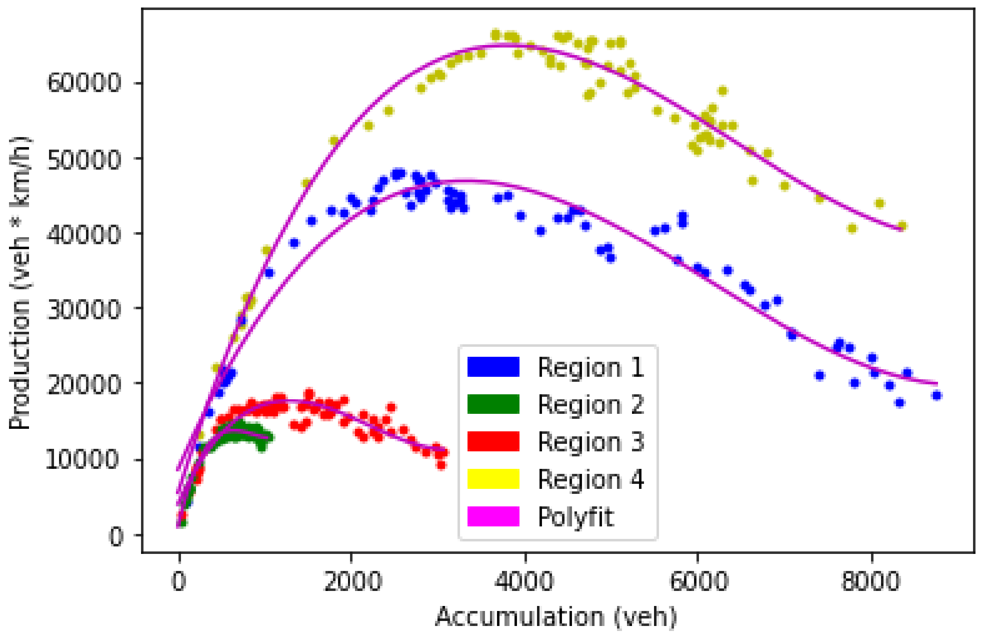


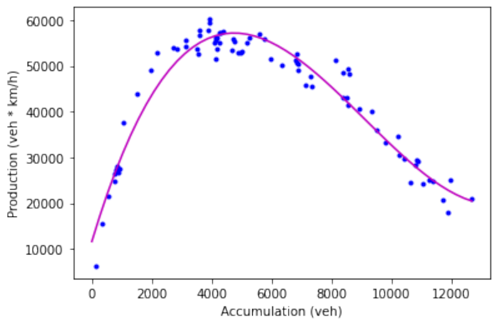
Figure 4: Production versus accumulation for each region in the network with 3rd degree polynomial curve fits superimposed.

The plot shown here is much more sensitive to the size of the regions in the network. A larger region can accumulate more vehicles as well as have a higher production. The previous MFD from Step 2 ignores this by utilizing volume and occupancy that are (more or less) independent of region size. The biggest difference is between regions 1 and 3. In Step 2, both of these regions followed a similar trend, however in this plot they are very different. This indicates that region 1 is much larger than region 3.

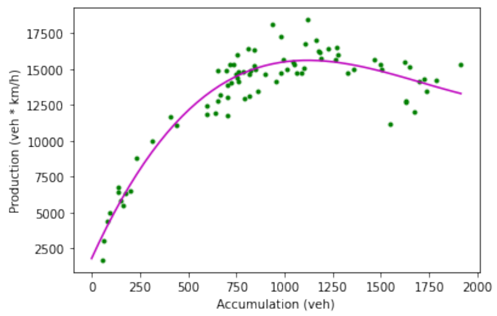
5 Step 5

For each region, first estimate a polynomial that fits the MFD found in Step 4. Then chose 50% of the links in each region according to the following three strategies: i) highest number of lanes; ii) longest link length; iii) maximum average flow. Produce production vs. accumulation plots and estimate the MFD of each region according to the proposed strategies (Note: scale both axes of the MFD by the ratio [total link length of the region/ total link length of the selected sample]). Finally, chose and compute a suitable error measurement to compare the MFDs obtained in this step with the corresponding ones of step 4.

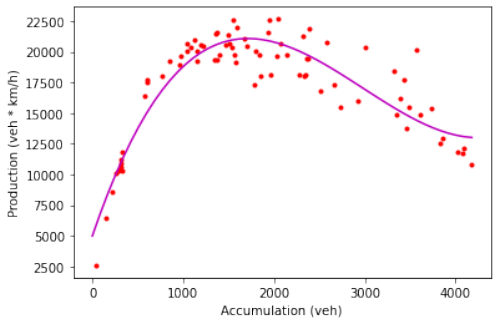
To simplify my analysis, I chose to calculate the average value for each of the specified criterion and only use links above the average. This may not be exactly 50% of the sample links, however it should be reasonably close.



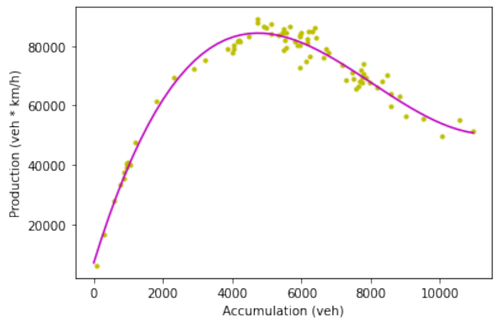
(a) Region 1



(b) Region 2

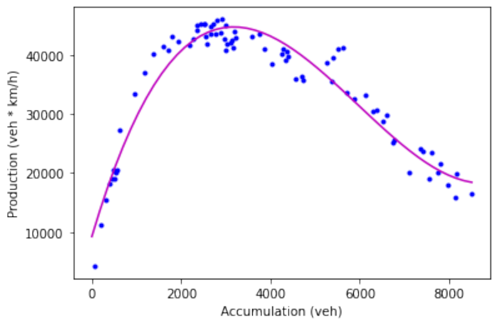


(c) Region 3

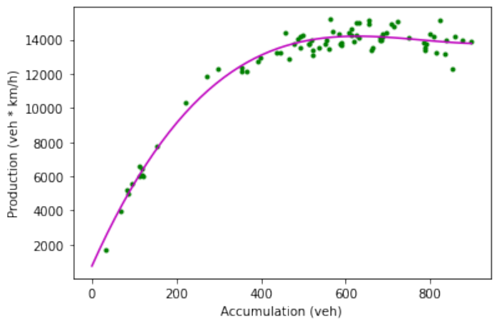


(d) Region 4

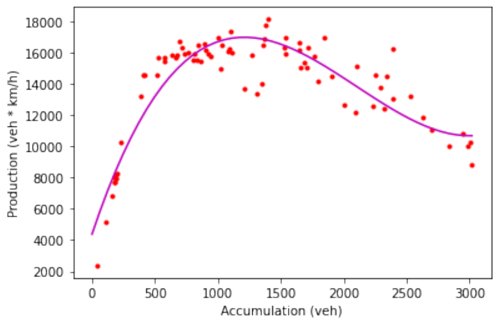
Figure 5: Production vs. accumulation for all four regions considering links with the highest number of lanes (i.e. above the average number of lanes for that region).



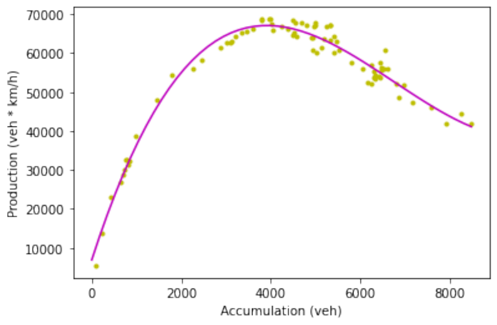
(a) Region 1



(b) Region 2

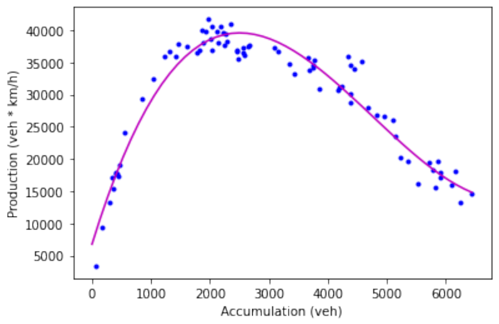


(c) Region 3

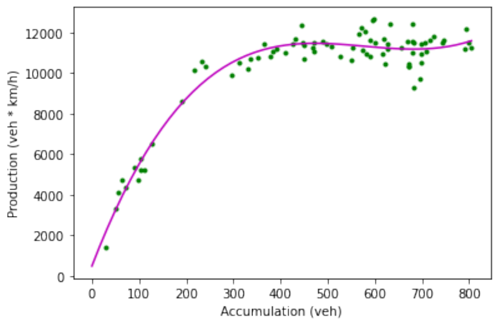


(d) Region 4

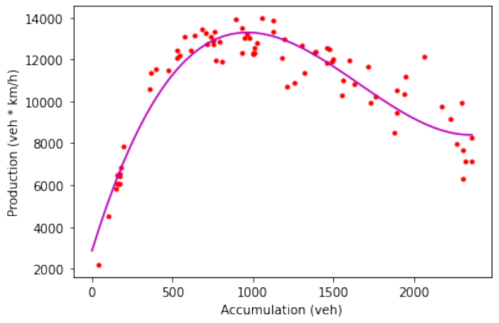
Figure 6: Production vs. accumulation for all four regions considering links with the highest link length (i.e. above the average link length for that region).



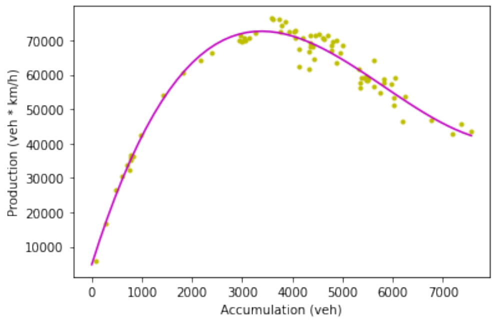
(a) Region 1



(b) Region 2

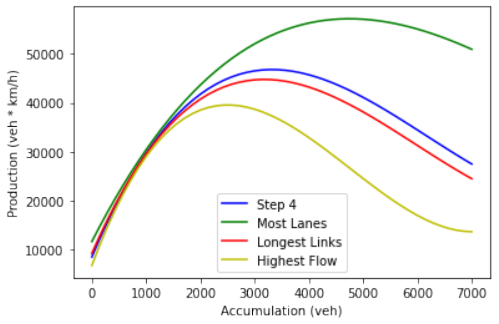


(c) Region 3

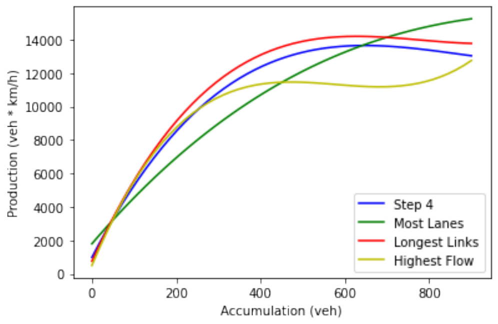


(d) Region 4

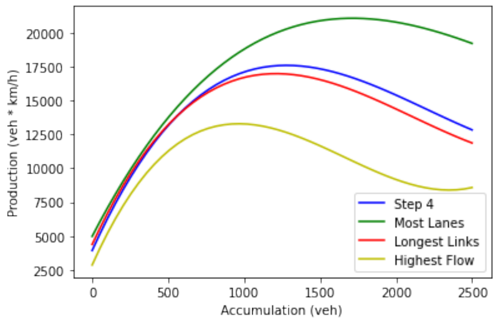
Figure 7: Production vs. accumulation for all four regions considering links with the highest flow (i.e. above the average flow for that region).



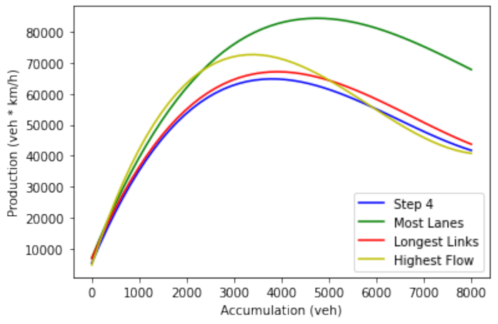
(a) Region 1



(b) Region 2



(c) Region 3



(d) Region 4

Figure 8: Trend lines from the previous parts are shown here for comparison.

It appears that by choosing the longest links we approximate the total region MFD better than with the other strategies. By choosing the links with the most lanes, we overestimate the production of the region. By choosing the links with the highest flow, we are potentially choosing the links that are prone to congestion and therefore do not accurately describe the region as a whole.

We could choose several methods to calculate the error as follows:

- The largest error (or difference) between the trend lines throughout the active accumulation region;
- The average error between the trend lines throughout the active accumulation region; or
- The error between the critical values on each trend line (i.e. the difference in critical accumulation values and maximum production values).

I would choose the average error (2nd bullet) as a preferred method as it more fully represents the difference in trends. Without performing any calculations, we can easily see that the 'Longest Links' trend line most accurately estimates the MFD from 'Step 4.'

6 Step 6

In the provided data file you have the measurements for all the links of the network. Assume now that for each region you want to select only 50% of the links and produce the production vs. accumulation MFD, similarly to Step 5. This time, you multiply all your measurements by a factor of 2 and then compare the resulting MFD with the one produced in Step 4 (that uses all the network links). Can you propose a methodology to select the 50% of the links so that the difference between the two MFDs is minimized?

Looking back to Step 4, we have the following equations.

$$\text{Accumulation} = \text{Density (veh/km)} \cdot \text{Link Length (km)} \tag{1}$$

$$\text{Production} = \text{Volume (veh/hr)} \cdot \text{Link Length (km)} \tag{2}$$

To understand the impact of doubling our measured values, we also need to review equations from Step 3:

$$\text{Density} \propto \frac{\text{Occupancy} \cdot \text{Lanes}}{\text{Link Length}} \tag{3}$$

The following values will be doubled:

- Occupancy;
- Lanes;
- Link Length; and
- Volume.

Therefore, we can see from equation 3 Density will be doubled. From equation 1, we can see that Accumulation will be quadrupled. Lastly, from equation 2, we can see that Production will also be quadrupled. This means that both Accumulation and Production will be scaled at about the same rate and the shape of our MFD should be similar to Step 4.

By choosing the links with the shortest Link Length (i.e. the bottom 50% of links based on their length) we should be able to minimize the error. This is because both Accumulation and Production depend on Link Length and by choosing the smallest values for this variable we will minimize the scaling of the new MFD.

7 Step 7

Looking at figure 1, we can see that the some regions contain links that are not connected to the main body of the region. I am unsure if this is intended or if it is an error in my program.

Overall, the level of occupancy in each region maintains a similar value over time. I would expect to set a threshold of variance in occupancy to perform the region clustering and attempt to keep clusters connected. We could step through neighboring links to determine their cluster based on average occupancy throughout the simulation time (utilizing breadth-first-search as mentioned in the course material).

Appendix: Full Jupyter Notebook

```
# %%
import csv
import numpy as np
import matplotlib.pyplot as plt

# Generate lists for our data.
nodes = []
links = []
flow = []
occupancy = []

with open('links.csv', encoding='utf-8-sig', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        links.append(row)
with open('nodes.csv', encoding='utf-8-sig', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
      nodes.append([np.int0(x) for x in row])
with open('occupancy.csv', encoding='utf-8-sig', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        occupancy.append([np.float16(x) for x in row])
with open('flow.csv', encoding='utf-8-sig', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        flow.append([np.int0(x) for x in row])

# %%
columns_n = list(zip(*nodes))
indices_n = columns_n[0]
indices_o = list(occupancy[0][:])

for i in range(len(occupancy[40][:]) - 1):
    color_val = occupancy[40][i+1] / 100.0
    i_1 = indices_n.index(np.int0(links[i][3]))
    i_2 = indices_n.index(np.int0(links[i][4]))
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '1':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='blue', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '2':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='green', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '3':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='red', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '4':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='yellow', alpha=color_val, linestyle='solid'
           )

# %%
for i in range(len(occupancy[60][:]) - 1):
    color_val = occupancy[60][i+1] / 100.0
    i_1 = indices_n.index(np.int0(links[i][3]))
    i_2 = indices_n.index(np.int0(links[i][4]))
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '1':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='blue', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '2':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='green', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '3':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='red', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '4':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='yellow', alpha=color_val, linestyle='solid'
           )

# %%
for i in range(len(occupancy[80][:]) - 1):
    color_val = occupancy[80][i+1] / 100.0
    i_1 = indices_n.index(np.int0(links[i][3]))
    i_2 = indices_n.index(np.int0(links[i][4]))
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '1':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='blue', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '2':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='green', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '3':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='red', alpha=color_val, linestyle='solid')
    if links[indices_o.index(occupancy[0][i+1]) - 1][5] == '4':
        plt.plot([nodes[i_1][1], nodes[i_2][1]], [nodes[i_1][2], nodes[
           i_2][2]], color='yellow', alpha=color_val, linestyle='solid'
           )

# %%
volume = flow
for i in range(len(volume)-1):
    for j in range(len(volume[i][0:-1])):
        volume[i+1][j+1] = volume[i+1][j+1] * 3600 / 90

# %%
for i in range(len(volume)-1):
    x = occupancy[i+1][3]
    y = volume[i+1][3]
    plt.plot(x, y, 'b.')
plt.ylabel('Volume (veh / hr)')
plt.xlabel('Occupancy (%)')

# %%
for i in range(len(volume)-1):
    x = (occupancy[i+1][3] + occupancy[i+1][2]) / 2
    y = (volume[i+1][3] + volume[i+1][2]) / 2
    plt.plot(x, y, 'b.')
plt.ylabel('Volume (veh / hr)')
plt.xlabel('Occupancy (%)')

# %%
for i in range(len(volume)-1):
    occ = 0
    vol = 0
    for j in range(len(volume[i][:])-1):
        occ += occupancy[i+1][j+1]
        vol += volume[i+1][j+1]
    occ /= len(occupancy[i][:]) - 1
    vol /= len(volume[i][:]) - 1
    plt.plot(occ, vol, 'b.')
plt.ylabel('Volume (veh / hr)')
plt.xlabel('Occupancy (%)')

# %%
import matplotlib.patches as mpatches

for i in range(len(volume)-1):
    occ1 = 0
    occ2 = 0
    occ3 = 0
    occ4 = 0
    vol1 = 0
    vol2 = 0
    vol3 = 0
    vol4 = 0
    count1 = 0
    count2 = 0
    count3 = 0
    count4 = 0
    for j in range(len(volume[i][:])-1):
        if links[indices_o.index(occupancy[0][j+1]) - 1][5] == '1':
            count1 += 1
            occ1 += occupancy[i+1][j+1]
            vol1 += volume[i+1][j+1]
        elif links[indices_o.index(occupancy[0][j+1]) - 1][5] == '2':
            count2 += 1
            occ2 += occupancy[i+1][j+1]
            vol2 += volume[i+1][j+1]
        elif links[indices_o.index(occupancy[0][j+1]) - 1][5] == '3':
            count3 += 1
            occ3 += occupancy[i+1][j+1]
            vol3 += volume[i+1][j+1]
        elif links[indices_o.index(occupancy[0][j+1]) - 1][5] == '4':
            count4 += 1
            occ4 += occupancy[i+1][j+1]
            vol4 += volume[i+1][j+1]
        else:
            print('no matches')
    occ1 /= count1
    vol1 /= count1
    occ2 /= count2
    vol2 /= count2
    occ3 /= count3
    vol3 /= count3
    occ4 /= count4
    vol4 /= count4
    plt.plot(occ1, vol1, 'b.', label='region 1')
    plt.plot(occ2, vol2, 'g.', label='region 2')
    plt.plot(occ3, vol3, 'r.', label='region 3')
    plt.plot(occ4, vol4, 'y.', label='region 4')
plt.ylabel('Volume (veh / hr)')
plt.xlabel('Occupancy (%)')
blue_patch = mpatches.Patch(color='blue', label='Region 1')
green_patch = mpatches.Patch(color='green', label='Region 2')
red_patch = mpatches.Patch(color='red', label='Region 3')
yellow_patch = mpatches.Patch(color='yellow', label='Region 4')
plt.legend(handles=[blue_patch, green_patch, red_patch, yellow_patch])

# %%
density = occupancy

for i in range(len(density) - 1):
    for j in range(len(density[i][:]) - 1):
        density[i+1][j+1] = density[i+1][j+1] / 100 * np.int0(links[
           indices_o.index(density[0][j+1]) - 1][2]) / 7 * 1000

# %%
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
for i in range(len(density)-1):
    den1 = 0
    den2 = 0
    den3 = 0
    den4 = 0
    vol1 = 0
    vol2 = 0
    vol3 = 0
    vol4 = 0
    count1 = 0
    count2 = 0
    count3 = 0
    count4 = 0
    for j in range(len(density[i][:])-1):
        laneLength = np.float16(links[indices_o.index(density[0][j+1])
           - 1][1]) / 1000
        if links[indices_o.index(density[0][j+1]) - 1][5] == '1':
            count1 += 1
            den1 += density[i+1][j+1] * laneLength
            vol1 += volume[i+1][j+1] * laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '2':
            count2 += 1
            den2 += density[i+1][j+1] * laneLength
            vol2 += volume[i+1][j+1] * laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '3':
            count3 += 1
```

```
                den3 += density[i+1][j+1] * laneLength
                vol3 += volume[i+1][j+1] * laneLength
            elif links[indices_o.index(density[0][j+1]) - 1][5] == '4':
                count4 += 1
                den4 += density[i+1][j+1] * laneLength
                vol4 += volume[i+1][j+1] * laneLength
            else:
                print('no matches')
        ax1.plot(i * 90, vol1 / den1, 'b.')
        ax1.plot(i * 90, vol2 / den2, 'g.')
        ax1.plot(i * 90, vol3 / den3, 'r.')
        ax1.plot(i * 90, vol4 / den4, 'y.')
        ax2.plot(den1 / count1, vol1 / den1, 'b.')
        ax2.plot(den2 / count2, vol2 / den2, 'g.')
        ax2.plot(den3 / count3, vol3 / den3, 'r.')
        ax2.plot(den4 / count4, vol4 / den4, 'y.')
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Mean Speed (km/h)')
ax2.set_xlabel('Density (veh/kn)')
ax2.set_ylabel('Mean Speed (km/h)')
ax1.legend(handles=[blue_patch, green_patch, red_patch, yellow_patch])
ax2.legend(handles=[blue_patch, green_patch, red_patch, yellow_patch])

# %%
r1a = []
r1p = []
r2a = []
r2p = []
r3a = []
r3p = []
r4a = []
r4p = []
for i in range(len(density)-1):
    acc1 = 0
    acc2 = 0
    acc3 = 0
    acc4 = 0
    pro1 = 0
    pro2 = 0
    pro3 = 0
    pro4 = 0
    for j in range(len(density[i][:])-1):
        laneLength = np.float16(links[indices_o.index(density[0][j+1])
            - 1][1]) / 1000
        if links[indices_o.index(density[0][j+1]) - 1][5] == '1':
            acc1 += density[i+1][j+1] * laneLength
            pro1 += volume[i+1][j+1] * laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '2':
                acc2 += density[i+1][j+1] * laneLength
                pro2 += volume[i+1][j+1] * laneLength
            elif links[indices_o.index(density[0][j+1]) - 1][5] == '3':
                acc3 += density[i+1][j+1] * laneLength
                pro3 += volume[i+1][j+1] * laneLength
            elif links[indices_o.index(density[0][j+1]) - 1][5] == '4':
                acc4 += density[i+1][j+1] * laneLength
                pro4 += volume[i+1][j+1] * laneLength
            else:
                print('no matches')
        plt.plot(acc1, pro1, 'b.')
        plt.plot(acc2, pro2, 'g.')
        plt.plot(acc3, pro3, 'r.')
        plt.plot(acc4, pro4, 'y.')
        r1a.append(acc1)
        r1p.append(pro1)
        r2a.append(acc2)
        r2p.append(pro2)
        r3a.append(acc3)
        r3p.append(pro3)
        r4a.append(acc4)
        r4p.append(pro4)
z1 = np.polyfit(r1a, r1p, 3)
z2 = np.polyfit(r2a, r2p, 3)
z3 = np.polyfit(r3a, r3p, 3)
z4 = np.polyfit(r4a, r4p, 3)
s4_trend1 = np.poly1d(z1)
s4_trend2 = np.poly1d(z2)
s4_trend3 = np.poly1d(z3)
s4_trend4 = np.poly1d(z4)
x1 = np.linspace(0, max(r1a))
x2 = np.linspace(0, max(r2a))
x3 = np.linspace(0, max(r3a))
x4 = np.linspace(0, max(r4a))
plt.plot(x1, s4_trend1(x1), 'm-')
plt.plot(x2, s4_trend2(x2), 'm-')
plt.plot(x3, s4_trend3(x3), 'm-')
plt.plot(x4, s4_trend4(x4), 'm-')
plt.xlabel('Accumulation (veh)')
plt.ylabel('Production (veh * km/h)')
magenta_patch = mpatches.Patch(color='magenta', label='Polyfit')
plt.legend(handles=[blue_patch, green_patch, red_patch, yellow_patch,
    magenta_patch])

# %%
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
fig3, ax3 = plt.subplots()
```

```
fig4, ax4 = plt.subplots()
numLanes1 = 0
count1 = 0
numLanes2 = 0
count2 = 0
numLanes3 = 0
count3 = 0
numLanes4 = 0
count4 = 0
linkLength1 = 0
linkLength2 = 0
linkLength3 = 0
linkLength4 = 0
for i in range(len(links)):
    if links[i][5] == '1':
        count1 += 1
        numLanes1 += np.int0(links[i][2])
        linkLength1 += np.float16(links[i][1]) / 1000
    if links[i][5] == '2':
        count2 += 1
        numLanes2 += np.int0(links[i][2])
        linkLength2 += np.float16(links[i][1]) / 1000
    if links[i][5] == '3':
        count3 += 1
        numLanes3 += np.int0(links[i][2])
        linkLength3 += np.float16(links[i][1]) / 1000
    if links[i][5] == '4':
        count4 += 1
        numLanes4 += np.int0(links[i][2])
        linkLength4 += np.float16(links[i][1]) / 1000
r1a = []
r1p = []
r2a = []
r2p = []
r3a = []
r3p = []
r4a = []
r4p = []
sampleLength1 = 0
sampleLength2 = 0
sampleLength3 = 0
sampleLength4 = 0
for i in range(len(density)-1):
    acc1 = 0
    acc2 = 0
    acc3 = 0
    acc4 = 0
    pro1 = 0
```

```
        pro2 = 0
        pro3 = 0
        pro4 = 0
        for j in range(len(density[i][:])-1):
            laneLength = np.float16(links[indices_o.index(density[0][j+1])
                 - 1][1]) / 1000
            if links[indices_o.index(density[0][j+1]) - 1][5] == '1' and np
                .int0(links[indices_o.index(density[0][j+1]) - 1][2]) >
                numLanes1 / count1:
                 acc1 += density[i+1][j+1] * laneLength
                 pro1 += volume[i+1][j+1] * laneLength
                 sampleLength1 += laneLength
            elif links[indices_o.index(density[0][j+1]) - 1][5] == '2' and
                np.int0(links[indices_o.index(density[0][j+1]) - 1][2]) >
                numLanes2 / count2:
                 acc2 += density[i+1][j+1] * laneLength
                 pro2 += volume[i+1][j+1] * laneLength
                 sampleLength2 += laneLength
            elif links[indices_o.index(density[0][j+1]) - 1][5] == '3' and
                np.int0(links[indices_o.index(density[0][j+1]) - 1][2]) >
                numLanes3 / count3:
                 acc3 += density[i+1][j+1] * laneLength
                 pro3 += volume[i+1][j+1] * laneLength
                 sampleLength3 += laneLength
            elif links[indices_o.index(density[0][j+1]) - 1][5] == '4' and
                np.int0(links[indices_o.index(density[0][j+1]) - 1][2]) >
                numLanes4 / count4:
                 acc4 += density[i+1][j+1] * laneLength
                 pro4 += volume[i+1][j+1] * laneLength
                 sampleLength4 += laneLength
        r1a.append(acc1)
        r1p.append(pro1)
        r2a.append(acc2)
        r2p.append(pro2)
        r3a.append(acc3)
        r3p.append(pro3)
        r4a.append(acc4)
        r4p.append(pro4)
    r1a = [x * linkLength1 / sampleLength1 * (len(density) - 1) for x in
        r1a]
    r2a = [x * linkLength2 / sampleLength2 * (len(density) - 1) for x in
        r2a]
    r3a = [x * linkLength3 / sampleLength3 * (len(density) - 1) for x in
        r3a]
    r4a = [x * linkLength4 / sampleLength4 * (len(density) - 1) for x in
        r4a]
    r1p = [x * linkLength1 / sampleLength1 * (len(density) - 1) for x in
        r1p]
```

```
r2p = [x * linkLength2 / sampleLength2 * (len(density) - 1) for x in
   r2p]
r3p = [x * linkLength3 / sampleLength3 * (len(density) - 1) for x in
   r3p]
r4p = [x * linkLength4 / sampleLength4 * (len(density) - 1) for x in
   r4p]
z1 = np.polyfit(r1a, r1p, 3)
z2 = np.polyfit(r2a, r2p, 3)
z3 = np.polyfit(r3a, r3p, 3)
z4 = np.polyfit(r4a, r4p, 3)
s5l_trend1 = np.poly1d(z1)
s5l_trend2 = np.poly1d(z2)
s5l_trend3 = np.poly1d(z3)
s5l_trend4 = np.poly1d(z4)
x1 = np.linspace(0, max(r1a))
x2 = np.linspace(0, max(r2a))
x3 = np.linspace(0, max(r3a))
x4 = np.linspace(0, max(r4a))
ax1.plot(r1a, r1p, 'b.', x1, s5l_trend1(x1), 'm-')
ax1.set_xlabel('Accumulation (veh)')
ax1.set_ylabel('Production (veh * km/h)')
ax2.plot(r2a, r2p, 'g.', x2, s5l_trend2(x2), 'm-')
ax2.set_xlabel('Accumulation (veh)')
ax2.set_ylabel('Production (veh * km/h)')
ax3.plot(r3a, r3p, 'r.', x3, s5l_trend3(x3), 'm-')
ax3.set_xlabel('Accumulation (veh)')
ax3.set_ylabel('Production (veh * km/h)')
ax4.plot(r4a, r4p, 'y.', x4, s5l_trend4(x4), 'm-')
ax4.set_xlabel('Accumulation (veh)')
ax4.set_ylabel('Production (veh * km/h)')

# %%
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
fig3, ax3 = plt.subplots()
fig4, ax4 = plt.subplots()
count1 = 0
count2 = 0
count3 = 0
count4 = 0
linkLength1 = 0
linkLength2 = 0
linkLength3 = 0
linkLength4 = 0
for i in range(len(links)):
    if links[i][5] == '1':
        count1 += 1
        linkLength1 += np.float16(links[i][1]) / 1000
    if links[i][5] == '2':
        count2 += 1
        linkLength2 += np.float16(links[i][1]) / 1000
    if links[i][5] == '3':
        count3 += 1
        linkLength3 += np.float16(links[i][1]) / 1000
    if links[i][5] == '4':
        count4 += 1
        linkLength4 += np.float16(links[i][1]) / 1000
r1a = []
r1p = []
r2a = []
r2p = []
r3a = []
r3p = []
r4a = []
r4p = []
sampleLength1 = 0
sampleLength2 = 0
sampleLength3 = 0
sampleLength4 = 0
for i in range(len(density)-1):
    acc1 = 0
    acc2 = 0
    acc3 = 0
    acc4 = 0
    pro1 = 0
    pro2 = 0
    pro3 = 0
    pro4 = 0
    for j in range(len(density[i][:])-1):
        laneLength = np.float16(links[indices_o.index(density[0][j+1])
        - 1][1]) / 1000
        if links[indices_o.index(density[0][j+1]) - 1][5] == '1' and
           laneLength > linkLength1 / count1:
        acc1 += density[i+1][j+1] * laneLength
        pro1 += volume[i+1][j+1] * laneLength
        sampleLength1 += laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '2' and
           laneLength > linkLength2 / count2:
        acc2 += density[i+1][j+1] * laneLength
        pro2 += volume[i+1][j+1] * laneLength
        sampleLength2 += laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '3' and
           laneLength > linkLength3 / count3:
        acc3 += density[i+1][j+1] * laneLength
        pro3 += volume[i+1][j+1] * laneLength
        sampleLength3 += laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '4' and
           laneLength > linkLength4 / count4:
        acc4 += density[i+1][j+1] * laneLength
        pro4 += volume[i+1][j+1] * laneLength
        sampleLength4 += laneLength
    r1a.append(acc1)
    r1p.append(pro1)
    r2a.append(acc2)
    r2p.append(pro2)
    r3a.append(acc3)
    r3p.append(pro3)
    r4a.append(acc4)
    r4p.append(pro4)
r1a = [x * linkLength1 / sampleLength1 * (len(density) - 1) for x in
    r1a]
r2a = [x * linkLength2 / sampleLength2 * (len(density) - 1) for x in
    r2a]
r3a = [x * linkLength3 / sampleLength3 * (len(density) - 1) for x in
    r3a]
r4a = [x * linkLength4 / sampleLength4 * (len(density) - 1) for x in
    r4a]
r1p = [x * linkLength1 / sampleLength1 * (len(density) - 1) for x in
    r1p]
r2p = [x * linkLength2 / sampleLength2 * (len(density) - 1) for x in
    r2p]
r3p = [x * linkLength3 / sampleLength3 * (len(density) - 1) for x in
    r3p]
r4p = [x * linkLength4 / sampleLength4 * (len(density) - 1) for x in
    r4p]
z1 = np.polyfit(r1a, r1p, 3)
z2 = np.polyfit(r2a, r2p, 3)
z3 = np.polyfit(r3a, r3p, 3)
z4 = np.polyfit(r4a, r4p, 3)
s5d_trend1 = np.poly1d(z1)
s5d_trend2 = np.poly1d(z2)
s5d_trend3 = np.poly1d(z3)
s5d_trend4 = np.poly1d(z4)
x1 = np.linspace(0, max(r1a))
x2 = np.linspace(0, max(r2a))
x3 = np.linspace(0, max(r3a))
x4 = np.linspace(0, max(r4a))
ax1.plot(r1a, r1p, 'b.', x1, s5d_trend1(x1), 'm-')
ax1.set_xlabel('Accumulation (veh)')
ax1.set_ylabel('Production (veh * km/h)')
ax2.plot(r2a, r2p, 'g.', x2, s5d_trend2(x2), 'm-')
ax2.set_xlabel('Accumulation (veh)')
ax2.set_ylabel('Production (veh * km/h)')
ax3.plot(r3a, r3p, 'r.', x3, s5d_trend3(x3), 'm-')
```

```
ax3.set_xlabel('Accumulation␣(veh)')
ax3.set_ylabel('Production␣(veh␣*␣km/h)')
ax4.plot(r4a, r4p, 'y.', x4, s5d_trend4(x4), 'm-')
ax4.set_xlabel('Accumulation␣(veh)')
ax4.set_ylabel('Production␣(veh␣*␣km/h)')

# %%
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
fig3, ax3 = plt.subplots()
fig4, ax4 = plt.subplots()
count1 = 0
count2 = 0
count3 = 0
count4 = 0
flow1 = 0
flow2 = 0
flow3 = 0
flow4 = 0
np_flow = np.array(flow)
avgFlows = np_flow.mean(axis=0)
for i in range(len(indices_o) - 1):
    if links[indices_o.index(density[0][i+1]) - 1][5] == '1':
        count1 += 1
        flow1 += avgFlows[i+1]
    if links[indices_o.index(density[0][i+1]) - 1][5] == '2':
        count2 += 1
        flow2 += avgFlows[i+1]
    if links[indices_o.index(density[0][i+1]) - 1][5] == '3':
        count3 += 1
        flow3 += avgFlows[i+1]
    if links[indices_o.index(density[0][i+1]) - 1][5] == '4':
        count4 += 1
        flow4 += avgFlows[i+1]
flow1 /= count1
flow2 /= count2
flow3 /= count3
flow4 /= count4
linkLength1 = 0
linkLength2 = 0
linkLength3 = 0
linkLength4 = 0
for i in range(len(density) - 1):
    for j in range(len(density[i][:]) - 1):
        if links[indices_o.index(density[0][j+1]) - 1][5] == '1':
            linkLength1 += np.float16(links[i][1]) / 1000
        if links[indices_o.index(density[0][j+1]) - 1][5] == '2':
            linkLength2 += np.float16(links[i][1]) / 1000
            if links[indices_o.index(density[0][j+1]) - 1][5] == '3':
                linkLength3 += np.float16(links[i][1]) / 1000
            if links[indices_o.index(density[0][j+1]) - 1][5] == '4':
                linkLength4 += np.float16(links[i][1]) / 1000
r1a = []
r1p = []
r2a = []
r2p = []
r3a = []
r3p = []
r4a = []
r4p = []
sampleLength1 = 0
sampleLength2 = 0
sampleLength3 = 0
sampleLength4 = 0
for i in range(len(density)-1):
    acc1 = 0
    acc2 = 0
    acc3 = 0
    acc4 = 0
    pro1 = 0
    pro2 = 0
    pro3 = 0
    pro4 = 0
    for j in range(len(density[i][:])-1):
        laneLength = np.float16(links[indices_o.index(density[0][j+1])
            - 1][1]) / 1000
        if links[indices_o.index(density[0][j+1]) - 1][5] == '1' and
            avgFlows[j] > flow1:
             acc1 += density[i+1][j+1] * laneLength
             pro1 += volume[i+1][j+1] * laneLength
             sampleLength1 += laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '2' and
            avgFlows[j] > flow2:
             acc2 += density[i+1][j+1] * laneLength
             pro2 += volume[i+1][j+1] * laneLength
             sampleLength2 += laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '3' and
            avgFlows[j] > flow3:
             acc3 += density[i+1][j+1] * laneLength
             pro3 += volume[i+1][j+1] * laneLength
             sampleLength3 += laneLength
        elif links[indices_o.index(density[0][j+1]) - 1][5] == '4' and
            avgFlows[j] > flow4:
             acc4 += density[i+1][j+1] * laneLength
             pro4 += volume[i+1][j+1] * laneLength
             sampleLength4 += laneLength
        r1a.append(acc1)
        r1p.append(pro1)
        r2a.append(acc2)
        r2p.append(pro2)
        r3a.append(acc3)
        r3p.append(pro3)
        r4a.append(acc4)
        r4p.append(pro4)
r1a = [x * linkLength1 / sampleLength1 for x in r1a]
r2a = [x * linkLength2 / sampleLength2 for x in r2a]
r3a = [x * linkLength3 / sampleLength3 for x in r3a]
r4a = [x * linkLength4 / sampleLength4 for x in r4a]
r1p = [x * linkLength1 / sampleLength1 for x in r1p]
r2p = [x * linkLength2 / sampleLength2 for x in r2p]
r3p = [x * linkLength3 / sampleLength3 for x in r3p]
r4p = [x * linkLength4 / sampleLength4 for x in r4p]
z1 = np.polyfit(r1a, r1p, 3)
z2 = np.polyfit(r2a, r2p, 3)
z3 = np.polyfit(r3a, r3p, 3)
z4 = np.polyfit(r4a, r4p, 3)
s5f_trend1 = np.poly1d(z1)
s5f_trend2 = np.poly1d(z2)
s5f_trend3 = np.poly1d(z3)
s5f_trend4 = np.poly1d(z4)
x1 = np.linspace(0, max(r1a))
x2 = np.linspace(0, max(r2a))
x3 = np.linspace(0, max(r3a))
x4 = np.linspace(0, max(r4a))
ax1.plot(r1a, r1p, 'b.', x1, s5f_trend1(x1), 'm-')
ax1.set_xlabel('Accumulation (veh)')
ax1.set_ylabel('Production (veh * km/h)')
ax2.plot(r2a, r2p, 'g.', x2, s5f_trend2(x2), 'm-')
ax2.set_xlabel('Accumulation (veh)')
ax2.set_ylabel('Production (veh * km/h)')
ax3.plot(r3a, r3p, 'r.', x3, s5f_trend3(x3), 'm-')
ax3.set_xlabel('Accumulation (veh)')
ax3.set_ylabel('Production (veh * km/h)')
ax4.plot(r4a, r4p, 'y.', x4, s5f_trend4(x4), 'm-')
ax4.set_xlabel('Accumulation (veh)')
ax4.set_ylabel('Production (veh * km/h)')

# %%
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()
fig3, ax3 = plt.subplots()
fig4, ax4 = plt.subplots()
x1 = np.linspace(0, 7000)
x2 = np.linspace(0, 900)
```

```
x3 = np.linspace(0, 2500)
x4 = np.linspace(0, 8000)
ax1.plot(x1, s4_trend1(x1), 'b-', label='Step 4')
ax1.plot(x1, s5l_trend1(x1), 'g-', label='Most Lanes')
ax1.plot(x1, s5d_trend1(x1), 'r-', label='Longest Links')
ax1.plot(x1, s5f_trend1(x1), 'y-', label='Highest Flow')
ax1.legend()
ax1.set_xlabel('Accumulation (veh)')
ax1.set_ylabel('Production (veh * km/h)')
ax2.plot(x2, s4_trend2(x2), 'b-', label='Step 4')
ax2.plot(x2, s5l_trend2(x2), 'g-', label='Most Lanes')
ax2.plot(x2, s5d_trend2(x2), 'r-', label='Longest Links')
ax2.plot(x2, s5f_trend2(x2), 'y-', label='Highest Flow')
ax2.legend()
ax2.set_xlabel('Accumulation (veh)')
ax2.set_ylabel('Production (veh * km/h)')
ax3.plot(x3, s4_trend3(x3), 'b-', label='Step 4')
ax3.plot(x3, s5l_trend3(x3), 'g-', label='Most Lanes')
ax3.plot(x3, s5d_trend3(x3), 'r-', label='Longest Links')
ax3.plot(x3, s5f_trend3(x3), 'y-', label='Highest Flow')
ax3.legend()
ax3.set_xlabel('Accumulation (veh)')
ax3.set_ylabel('Production (veh * km/h)')
ax4.plot(x4, s4_trend4(x4), 'b-', label='Step 4')
ax4.plot(x4, s5l_trend4(x4), 'g-', label='Most Lanes')
ax4.plot(x4, s5d_trend4(x4), 'r-', label='Longest Links')
ax4.plot(x4, s5f_trend4(x4), 'y-', label='Highest Flow')
ax4.legend()
ax4.set_xlabel('Accumulation (veh)')
ax4.set_ylabel('Production (veh * km/h)')
```