CSE 6730 Project Checkpoint

Traffic flow simulation: case study of Atlanta traffic

Team members: Shu Bin, Qinyu Wang, Zijian Li

Jupyter notebook environment setup

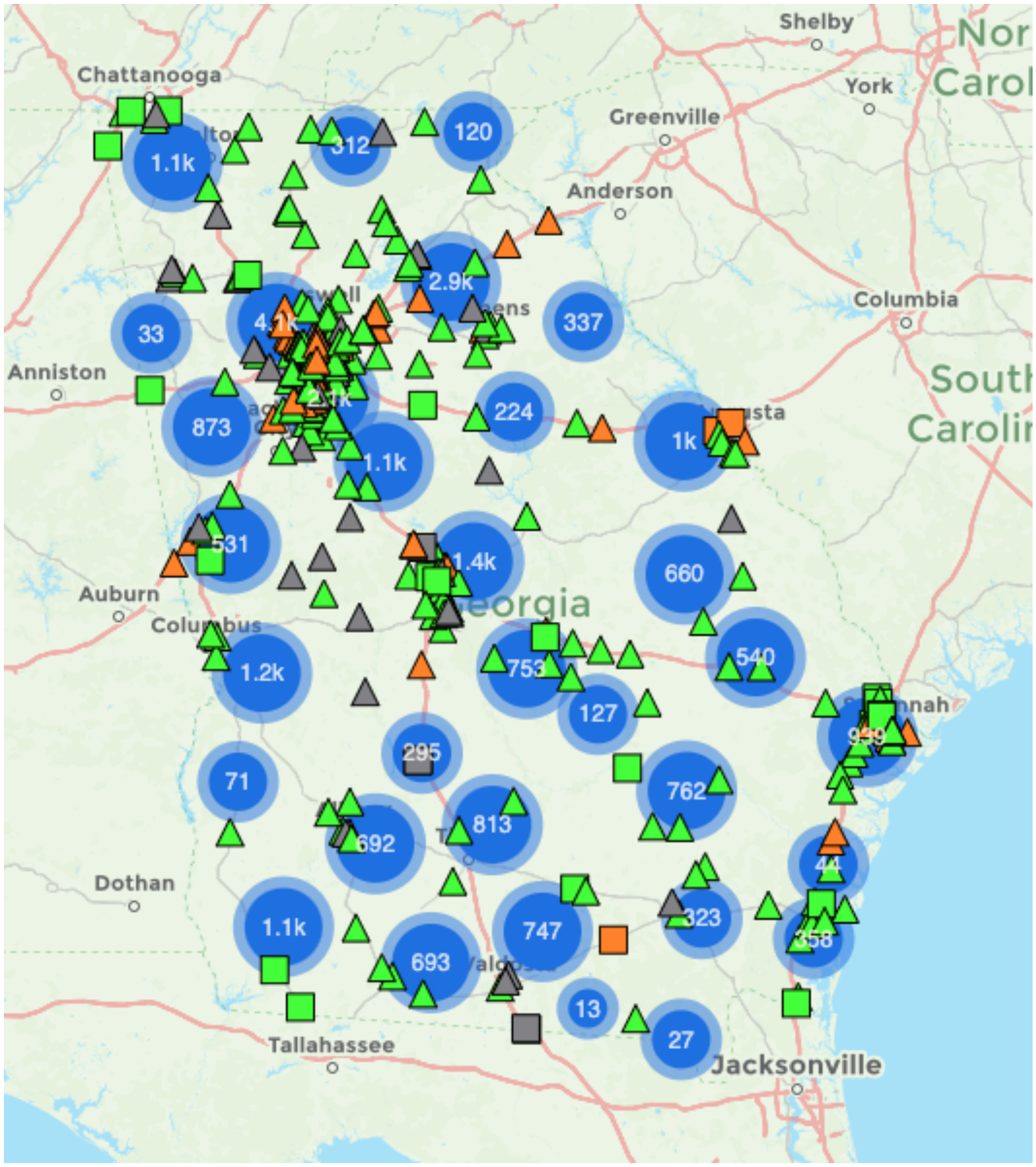
In [1]:
```
# Import packages used in this notebook
import math
import time
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
from IPython.display import Image, display
```

1. Introduction

The project objective is to build a mathematical model to simulate the traffic flow in Atlanta. The ultimate goal is to understand the traffic phenomena in order to optimize traffic networks, alleviate congestion, maximize traffic flow and decrease accident rate. To validify the reliability of our models, we will perform case studies via simulating the real traffic data in Atlanta with our models. The implemented models would be tweaked based on the validation.

2. Data collection

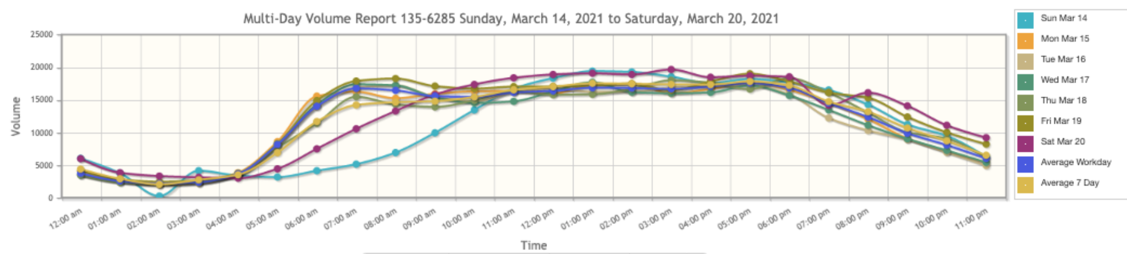
The Atlanta traffic data is collected from the website of Georgia Department of Transportation (GDOT): https://gdottrafficdata.drakewell.com/publicmultinodemap.asp (https://gdottrafficdata.drakewell.com/publicmultinodemap.asp)



The green triangles symbolize active continuous count station (CCS) which are the sources in this project for collecting the traffic data. For active CCS, the hourly real-time traffic flow (volume) and speed data are monitored. An example site data of weekly (03/14/2021 - 03/20/2021) volume and speed for station 135-6285 - I-85 of Jimmy Carter Blvd @Graves Rd NW, ATL is shown below:

Multi-Day Volume Report 135-6285 Sunday, March 14, 2021 to Saturday, March 20, 2021

≡ Menu X



Switch chart type | Open in new window | Export as image | Hide graph

Site Name: 135-6285 Site ID: 000001356285 Description: I-85 W of Jimmy Carter Blvd @Graves Rd NW, ATL

All Lanes ▾ | Time Period: 1 hour ▾ | Class: Any ▾ | Exclude data: None ▾

	Sun Mar 14	Mon Mar 15	Tue Mar 16	Wed Mar 17	Thu Mar 18	Fri Mar 19	Sat Mar 20	Average Workday	Average 7 Day	Total Count
12:00 am	6096	4174	3553	3553	3575	4188	6054	3809	4456	31193
01:00 am	3834	2780	2509	2480	2426	2883	3905	2616	2974	20817
02:00 am	324	2238	1978	2219	2181	2507	3384	2225	2119	14831
03:00 am	4205	2310	2296	2447	2456	2697	3198	2441	2801	19609
04:00 am	3460	3887	3577	3713	3608	3802	3061	3717	3587	25108
05:00 am	3234	8661	8072	8331	7717	8283	4501	8213	6971	48799
06:00 am	4201	15641	14007	14339	11472	14972	7561	14086	11742	82193
07:00 am	5205	16326	16903	17454	15558	17962	10632	16841	14291	100040
08:00 am	6992	15326	17300	17303	14379	18351	13366	16532	14717	103017
09:00 am	10025	16009	15544	15453	14001	17181	15885	15638	14871	104098
10:00 am	13528	16389	14852	14609	14990	16802	17443	15528	15516	108613
11:00 am	16778	16433	16255	14848	16384	17100	18455	16204	16608	116253
12:00 pm	18416	16059	16840	16374	15870	17126	18972	16454	17094	119657
01:00 pm	19484	17381	16296	17782	15933	17255	19154	16929	17612	123285
02:00 pm	19362	17515	16294	16260	16763	17553	18966	16877	17530	122713
03:00 pm	18636	16089	16951	16098	18069	16540	19732	16749	17445	122115
04:00 pm	17636	16729	17138	16209	17720	17892	18548	17138	17410	121872
05:00 pm	18286	17781	17372	17285	16711	19075	18768	17645	17897	125278
06:00 pm	17749	16603	15914	15746	18397	17911	18609	16914	17276	120929
07:00 pm	16586	14596	12293	13531	16199	16176	14237	14559	14803	103618
08:00 pm	14360	12123	10386	11185	13115	15392	16164	12440	13246	92725
09:00 pm	11308	9156	8976	9101	10101	12461	14145	9959	10750	75248
10:00 pm	9549	7139	7024	7319	9130	10094	11171	8141	8775	61426
11:00 pm	6348	5486	5004	5488	6148	8284	9275	6082	6576	46033
7am-7pm	182097	198640	197659	195421	194775	210748	208530	199449	198267	1387870
6am-10pm	228552	250156	243321	243577	245662	269749	260637	250493	248808	1741654
6am-12am	244449	262781	255349	256384	260940	288127	281083	264716	264159	1849113
12am-12am	265602	286831	277334	279127	282903	312487	305186	287736	287067	2009470
am Peak	11:00 am	11:00 am	08:00 am	07:00 am	11:00 am	08:00 am	11:00 am	07:00 am	11:00 am	
Peak Volume	16778	16433	17300	17454	16384	18351	18455	16841	16608	
pm Peak	01:00 pm	05:00 pm	05:00 pm	01:00 pm	06:00 pm	05:00 pm	03:00 pm	05:00 pm	05:00 pm	
Peak Volume	19484	17781	17372	17782	18397	19075	19732	17645	17897	

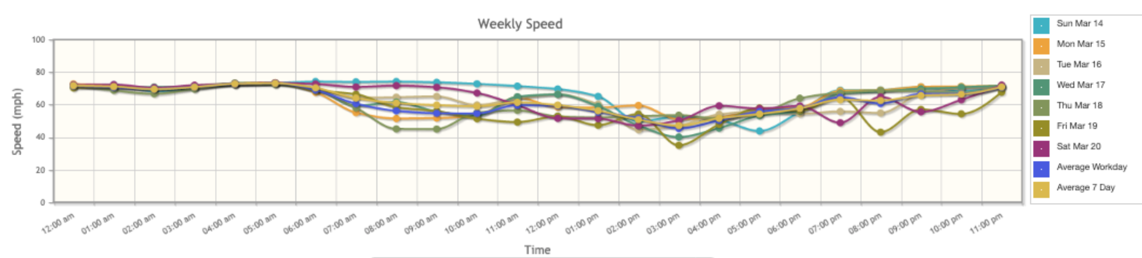
Event key: QC failure Atypical (QC) Events Special Holiday Offline
Weekends and defined holidays

Notes on data:
Weekly (7-day) averages are weighted by each day of the week.

Holidays & Events:
None

Weekly Speed Report 135-6285 Sunday, March 14, 2021 to Saturday, March 20, 2021

≡ Menu X



Switch chart type | Open in new window | Export as image | Hide graph

Site Name: 135-6285 Site ID: 000001356285 Description: I-85 W of Jimmy Carter Blvd @Graves Rd NW, ATL

All Lanes ▾ | Time Period: 1 hour ▾ | Class: Any ▾ | Speed units: mph ▾ | Exclude data: None ▾

	Sun Mar 14	Mon Mar 15	Tue Mar 16	Wed Mar 17	Thu Mar 18	Fri Mar 19	Sat Mar 20	Average Workday	Average 7 Day
12:00 am	72.3	73.2	72.0	71.6	72.6	70.8	72.6	72.0	72.2
01:00 am	72.6	72.2	71.3	71.2	69.3	71.4	72.8	71.1	71.6
02:00 am	68.8	70.6	70.8	71.3	67.2	70.3	70.9	70.0	70.0
03:00 am	70.4	71.4	71.2	71.8	70.1	72.1	72.4	71.3	71.3
04:00 am	73.7	73.7	73.0	73.6	72.6	73.2	73.3	73.2	73.3
05:00 am	73.8	74.0	73.7	73.6	72.8	73.6	73.8	73.6	73.6
06:00 am	74.7	68.1	70.7	70.1	69.3	69.2	73.2	69.5	70.8
07:00 am	74.5	55.7	64.8	59.8	58.5	66.9	71.4	61.1	64.5
08:00 am	74.7	52.0	65.1	62.1	45.6	58.9	72.2	56.7	61.5
09:00 am	74.2	52.3	65.6	56.3	45.5	56.5	71.2	55.2	60.2
10:00 am	73.3	53.8	60.3	53.8	55.6	51.9	67.7	55.1	59.5
11:00 am	71.9	64.9	64.3	65.4	57.0	49.8	60.2	60.2	61.9
12:00 pm	70.1	58.9	66.8	66.8	53.3	53.3	52.1	59.8	60.2
01:00 pm	65.6	58.8	60.7	59.5	52.7	47.9	51.9	55.9	56.7
02:00 pm	50.3	60.0	45.1	47.7	53.4	54.7	47.6	52.2	51.3
03:00 pm	53.1	51.6	47.7	40.6	54.0	35.5	50.8	45.9	47.6
04:00 pm	51.0	52.8	54.5	46.1	52.0	48.4	59.8	50.8	52.1
05:00 pm	44.3	57.3	56.4	53.7	55.1	58.4	58.1	56.2	54.7
06:00 pm	56.1	58.7	54.9	56.6	64.4	56.8	59.7	58.3	58.2
07:00 pm	69.4	69.0	56.3	67.1	67.8	65.3	49.4	65.1	63.5
08:00 pm	69.5	69.4	55.4	68.7	69.0	43.6	65.3	61.2	63.0
09:00 pm	71.3	71.5	67.2	70.2	68.8	57.6	56.1	67.1	66.1
10:00 pm	68.8	71.7	70.6	71.2	67.8	54.8	63.5	67.2	66.9
11:00 pm	71.2	72.1	71.2	72.1	72.0	68.2	72.5	71.1	71.4
Average	64.0	60.8	61.1	60.1	59.0	56.4	60.9	62.5	63.4
Maximum	74.7	74.0	73.7	73.6	72.8	73.6	73.8	73.6	73.6
Minimum	44.3	51.6	45.1	40.6	45.5	35.5	47.6	45.9	47.6

Event key: QC failure Atypical (QC) Events Special Holiday Offline
Weekends and defined holidays

Notes on data:
Weekly (7-day) averages are weighted by each day of the week.

Holidays & Events:
None

Data prepared by Drakewell US 01N - Nevada April 15, 2021 6:57:08 AM.

C2 > GDOT_CCS > Sites > 000001356285 (135-6285) > Weekly Speed Report 135-6285 Sunday, March 14, 2021 to Saturday, March 20, 2021

We compile the March 2021 traffic flow and speed data of four sites on the I-85 road into .csv files which are used in the modeling simulation and verification.

- **Station 1**: I-85 at North Druid Hills Road (station ID: 089-3323)
- **Station 2**: I-85/SR403 bn I-285 & Chamblee Tucker Rd, ATL (station ID: 089-3332)
- **Station 3**: I-85 W of Jimmy Carter Blvd @Graves Rd NW, ATL (station ID: 135-6285)
- **Station 4**: I-85 btwn Jimmy Carter & Indian Trail, Norcross (station ID: 135-6287)

3. Method and results

We intend to simulate the traffic flow with two models (progressively increasing in complexity). The first model would simulate the traffic flow with linear partial differential equations (PDE). This model simplifies the discrete traffic flow as continuous vehicle density. We would solve the PDEs analitically with and without assuming constant density and compare with the results derived from the iterative approach (Runge-Kutta 4th order). The second model (non-linear) would focus on more realistic conditions via taking into account more complex effects, such as the traffic lights, intersections, entrances and exits, etc.

3.1 Linear PDE model

In the linear PDE model, the traffic flow (the number of passing cars per unit time) is formulated as a function of position and time given the initial traffic density (the number of passing cars per unit length). The variables included in this model are

- **traffic flow**: J
- **position**: x
- **time**: t
- **vehicle density**: ρ
- **vehicle velocity**: v

For the purpose of simulating the traffic flow on a first-order, we make the following assumptions:

1. The vehicle density is continuous. It means that our approach does not use individual car behavior to implement the traffic flow simulation.
2. The vehicle motion is unidirectional from left to right on a one-lane road of infinite length.
3. All vehicles are assumed to have the same length (L) and they are evenly spaced at distance d, which is simplified as a uniform distribution model (the vehicle density is homogeneous along the infinite lane).
4. All other factors are not considered in the basic implementation of the model, but they might be tested and discussed given the progress and availability of the schedule.

3.1.1 Linear PDE formulation

The vehicle density ρ is formulated as a function of position x and time t. The vehicle density at position x_0 and time t_0 is calculated as follows according to the definition.

$$\rho(x_0, t_0) = \frac{1}{L+d} \qquad \text{(Eq. 1)}$$

Given the non-negativity of vehicle spacing d, the traffic density is upper bounded by $\frac{1}{L}$.
The traffic flow J can be expressed as a linear combination of vehicle speed and traffic density as follows

$$J(x_0, t_0) = \rho(x_0, t_0) * v(x_0, t_0) \qquad \text{(Eq. 2)}$$

Substitute Eq. 1 in Eq. 2, we get

$$J(x_0, t_0) = \frac{v(x_0,t_0)}{L+d} \qquad \text{(Eq. 3)}$$

Consider a position interval Δx and time interval Δt around x_0 and t_0, according to the *Balance law* for traffic density, the following equation can be derived

$$\Delta x * (\rho(x_0, t_0 + \Delta t) - \rho(x_0, t_0)) = \Delta t * (J(x_0 + \Delta x, t_0) - J(x_0, t_0)) \qquad \text{(Eq. 4)}$$

After conducting Taylor expansion on both sides of Eq. 4 and omitting the third-order in terms of Δx and Δt, we get

$$\frac{\partial \rho}{\partial t} = -\frac{\partial J}{\partial x} \qquad \text{(Eq. 5)}$$

According to Eq.2, substitute J with ρv, the linear PDE model for traffic flow is

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v}{\partial x} = 0 \qquad \text{(Eq. 6)}$$
$$\frac{\partial \rho}{\partial t} + \rho\frac{\partial v}{\partial x} + v\frac{\partial \rho}{\partial x} = 0 \qquad \text{(Eq. 7)}$$

Eq. 6 (or Eq. 7) is the expression of continuous traffic flow equation.
Since there are two scenarios of traffic density (spatially homogeneous and spatially heterogenous) between adjacent stations in real traffic, they are separately simulated and discussed below.

3.1.2 Spatially homogeneous traffic density

If there is no intersections between adjacent stations, the traffic densities of these two stations are normally similar with small discrepancy. An example of this scenario is illustrated below. We extract the daily traffic flow and speed data (**March 15, 2021, Monday**) of two adjacent stations on the I-85 road without intersections in between. The information of these two stations are as follows.

- **Entrance (Station 3)**: I-85 W of Jimmy Carter Blvd @Graves Rd NW, ATL (station ID: 135-6285)
- **Exit (Station 4)**: I-85 btwn Jimmy Carter & Indian Trail, Norcross (station ID: 135-6287)

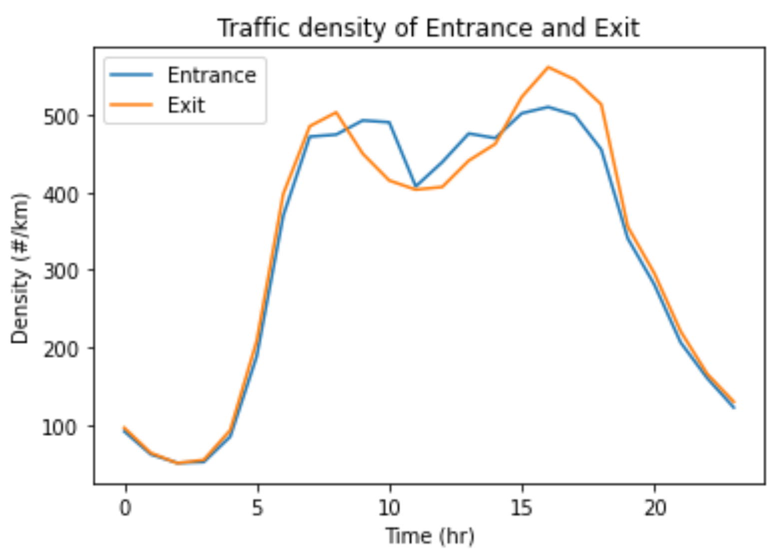
The traffic density at time t_n at location x_i is calculated as the result of traffic flow dividing speed.

$$\rho(x_i, t_n) = \frac{J(x_i,t_n)}{v(x_i,t_n)} \qquad \text{(Eq. 8)}$$

The comparision of traffic density on March 15, 2021 of the entrance and the exit is shown below.

In [2]:

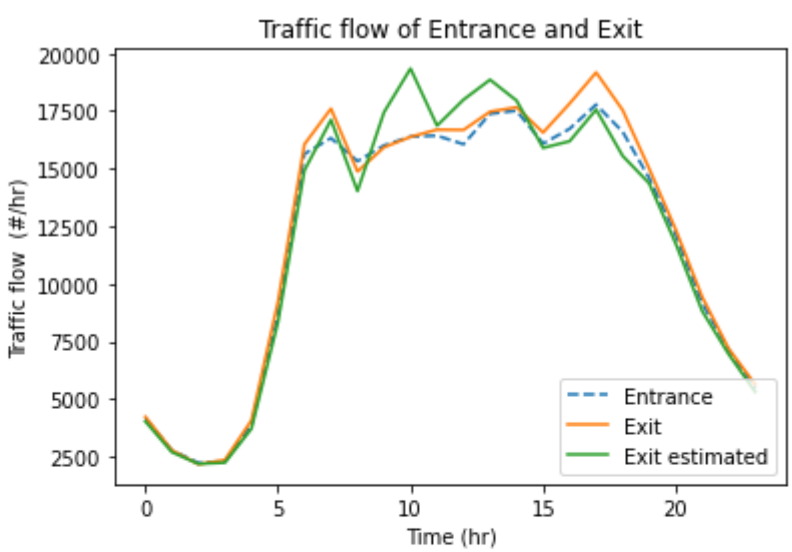
```
# Read .csv file and extract column data
col_name_val1 = ['entrance_volume', 'entrance_speed', 'exit_volume', 'e
data_val1 = pd.read_csv('gdot_1356285_1356287_031521_firstValidation.cs
entrance_volume = data_val1.loc[1:,'entrance_volume'].tolist()
exit_volume = data_val1.loc[1:,'exit_volume'].tolist()
entrance_speed = data_val1.loc[1:,'entrance_speed'].tolist()
exit_speed = data_val1.loc[1:,'exit_speed'].tolist()
time_val1 = data_val1.loc[1:,'time'].tolist()
flow_data_size_val1 = len(entrance_volume)

# Convert data from format string to float
for i in range(flow_data_size_val1):
    entrance_volume[i] = float(entrance_volume[i])
    exit_volume[i] = float(exit_volume[i])
    entrance_speed[i] = float(entrance_speed[i])
    exit_speed[i] = float(exit_speed[i])
    time_val1[i] = float(time_val1[i])

# Calculate traffic density
entrance_density = np.zeros(flow_data_size_val1)
exit_density = np.zeros(flow_data_size_val1)
for i in range(flow_data_size_val1):
  entrance_density[i] = entrance_volume[i] / entrance_speed[i]
  exit_density[i] = exit_volume[i] / exit_speed[i]

# Plot traffic density of entrance and exit
plt.plot(time_val1, entrance_density*1.60934, label='Entrance')
plt.plot(time_val1, exit_density*1.60934, label='Exit')
plt.title('Traffic density of Entrance and Exit')
plt.xlabel('Time (hr)')
plt.ylabel('Density (#/km)')
plt.legend(loc= 'upper left')
plt.show()
```



The result shows that the traffic density at the two stations generally approximate each other although some variations exit during the day. It demonstrates that the traffic density is roughly spatially homogeneous if there is no intersections. With this assumption, the traffic flow of a targeted location can be estimated by the traffic density of another location and the speed of the targeted location. For instance, we estimate the traffic flow of the exit by leveraging the traffic density of the entrace as follows.

In [3]:

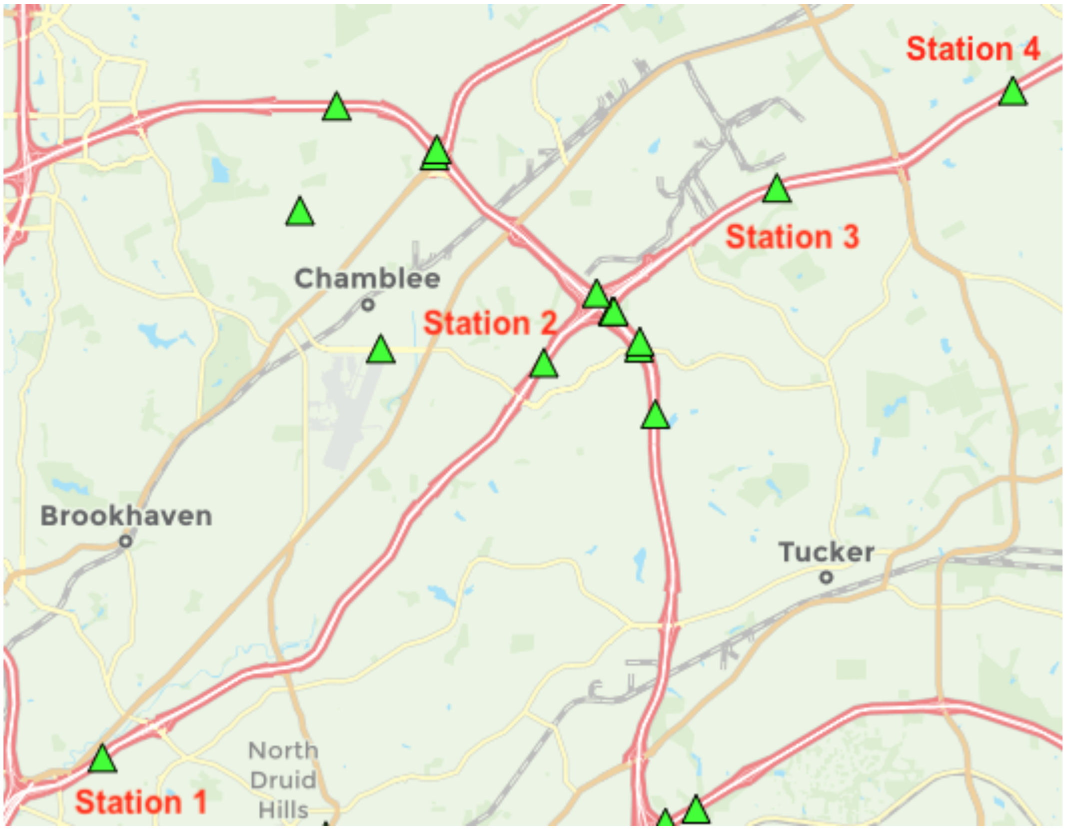
```
# Estimate exit volume based on entrance density and exit speed
exit_volume_estimated = np.zeros(flow_data_size_val1)
for i in range(flow_data_size_val1):
    exit_volume_estimated[i] = entrance_density[i] * exit_speed[i]

# Calculate absolute error between estimated traffic flow and true valu
# entrance_exit_abs_error = np.zeros(flow_data_size_val1)
# for i in range(flow_data_size_val1):
#   entrance_exit_abs_error[i] = 100 * abs((exit_volume_estimated[i]-ex

# Plot comparison of entrance and exit traffic flow (volume) data
plt.plot(time_val1, entrance_volume, '--', label='Entrance')
plt.plot(time_val1, exit_volume, label='Exit')
plt.plot(time_val1, exit_volume_estimated, label='Exit estimated')
plt.title('Traffic flow of Entrance and Exit')
plt.xlabel('Time (hr)')
plt.ylabel('Traffic flow  (#/hr)')
plt.legend(loc= 'lower right')
plt.show()

# Plot absolute error
# plt.plot(time_val1, entrance_exit_abs_error)
# plt.title('Relative percentage error between Entrance & Exit')
# plt.xlabel('Time (hr)')
# plt.ylabel('Percentage error (%)')
# plt.show()
```



The above plot shows that during the time slot that the two stations have similar traffic density, the estimated traffic flow approximates the true traffic flow very well. However, the error of the estimation increases for the time window where relatively large discrepancy of traffic density occurs.

3.1.3 Spatially heterogeneous traffic density

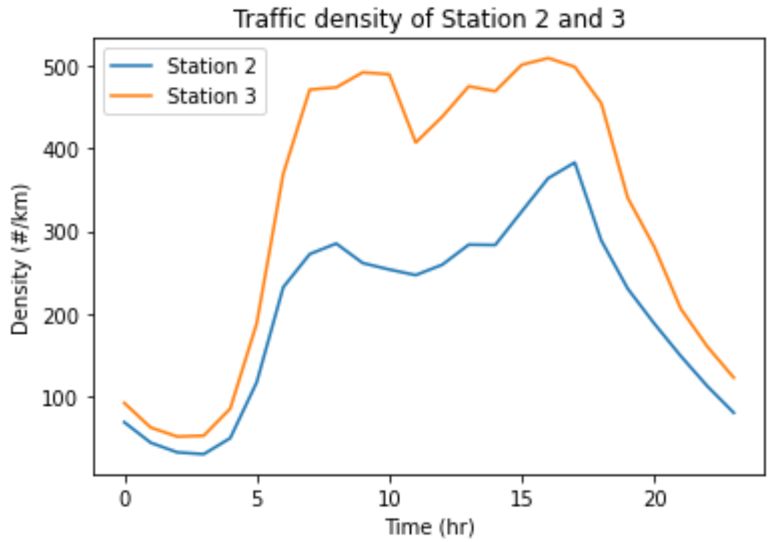
Another scenario is that the traffic densities of the adjacent stations are different. This case normally appears when there are intersections between the stations. The potential imbalance between the input and output traffic flux at the intersections may cause the difference in the traffic density. We extract the daily (**March 15, 2021, Monday**) traffic data of a pair of adjacent stations (Station 2 and 3) with an intersection in between to illustrate this case. The geographic locations of Station 2 and 3 together with the two boundaries Station 1 and 4 (used in the simulation) are shown below.



The comparison of traffic density between Station 2 and 3 is shown below.

In [4]:

```
# Read .csv file and extract column data
col_name_val2 = ['volume_S1', 'speed_S1', 'volume_S2', 'speed_S2','volu
data_val2 = pd.read_csv('gdot_0893323_0893332_1356285_1356287_031521_se
volume_S1 = data_val2.loc[1:,'volume_S1'].tolist()
volume_S2 = data_val2.loc[1:,'volume_S2'].tolist()
volume_S3 = data_val2.loc[1:,'volume_S3'].tolist()
volume_S4 = data_val2.loc[1:,'volume_S4'].tolist()
speed_S1 = data_val2.loc[1:,'speed_S1'].tolist()
speed_S2 = data_val2.loc[1:,'speed_S2'].tolist()
speed_S3 = data_val2.loc[1:,'speed_S3'].tolist()
speed_S4 = data_val2.loc[1:,'speed_S4'].tolist()
time_val2 = data_val2.loc[1:,'time'].tolist()
flow_data_size_val2 = len(volume_S1)

# Convert data from format string to float
for i in range(flow_data_size_val2):
    volume_S1[i] = float(volume_S1[i])
    volume_S2[i] = float(volume_S2[i])
    volume_S3[i] = float(volume_S3[i])
    volume_S4[i] = float(volume_S4[i])
    speed_S1[i] = float(speed_S1[i])
    speed_S2[i] = float(speed_S2[i])
    speed_S3[i] = float(speed_S3[i])
    speed_S4[i] = float(speed_S4[i])
    time_val2[i] = float(time_val2[i])

# Calculate density for each station & density_max and speed_max from S
density_S1 = np.zeros(flow_data_size_val2)
density_S2 = np.zeros(flow_data_size_val2)
density_S3 = np.zeros(flow_data_size_val2)
density_S4 = np.zeros(flow_data_size_val2)
density_max = np.zeros(flow_data_size_val2)
speed_max = np.zeros(flow_data_size_val2)

for i in range(flow_data_size_val2):
  density_S1[i] = volume_S1[i] / speed_S1[i]
  density_S2[i] = volume_S2[i] / speed_S2[i]
  density_S3[i] = volume_S3[i] / speed_S3[i]
  density_S4[i] = volume_S4[i] / speed_S4[i]
  density_max[i] = (speed_S1[i]*density_S4[i] - speed_S4[i]*density_S1[
  speed_max[i] = speed_S1[i] / (1 - density_S1[i]/density_max[i])

# Plot density of Station 2 and 3
plt.plot(time_val2, density_S2*1.60934, label='Station 2')
plt.plot(time_val2, density_S3*1.60934, label='Station 3')
plt.title('Traffic density of Station 2 and 3')
plt.xlabel('Time (hr)')
plt.ylabel('Density (#/km)')
plt.legend(loc= 'upper left')
plt.show()
```



The above plot shows that there are large difference between the Station 2 and 3, especially during the day time. Hence, the assumption of spatially homogeneous traffic density no longer holds, and it is expected that to estimate the traffic flow of Station 3 via the traffic density of Station 2 would induce large errors.

The idea is to explore the correlation between the speed and the traffic density from the traffic data of Station 1 and 4 and then use this derived correlation to fit the traffic flow given the density data for Station 2 and 3.

Two approaches have been tested. The first approach is to assume a linear velocity formulated by the **Greenshields** model as follows.

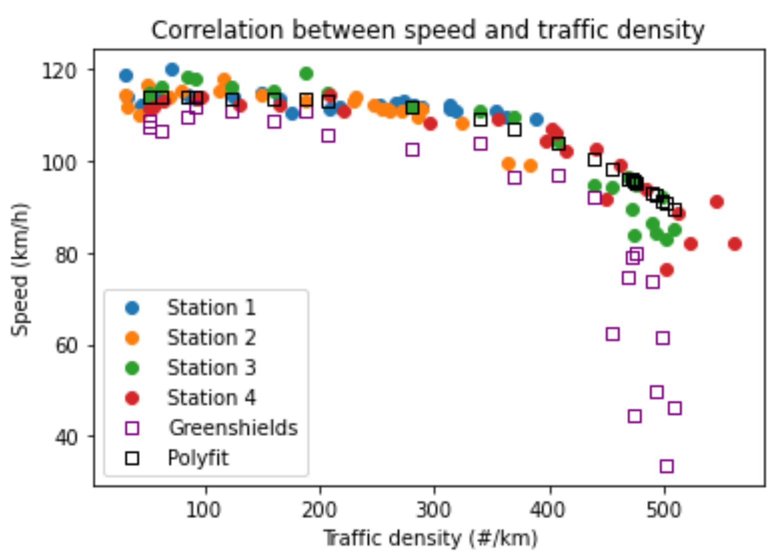
$$v = v_{max}(1 - \frac{\rho}{\rho_{max}}) \qquad \text{(Eq. 9)}$$

where v_{max} and ρ_{max} are two unknown parameters (for each hour) that can be calculated by plugging hourly traffic data of Station 1 and 4. Then with derived parameters, the estimated traffic flow for Station 2 and 3 can be calculated as the product of true traffic density and estimated speed

$$J = \rho v_{max}(1 - \frac{\rho}{\rho_{max}}) \qquad \text{(Eq. 10)}$$

The second approach is to use **polynomial regression** to fit the correlation. The true corrlations between the traffic density and speed for all four stations and the fitted correlations derived from Greenshields model and polynomial regression (fitted with the traffic data of Station 3) are shown in the scatter plot below.

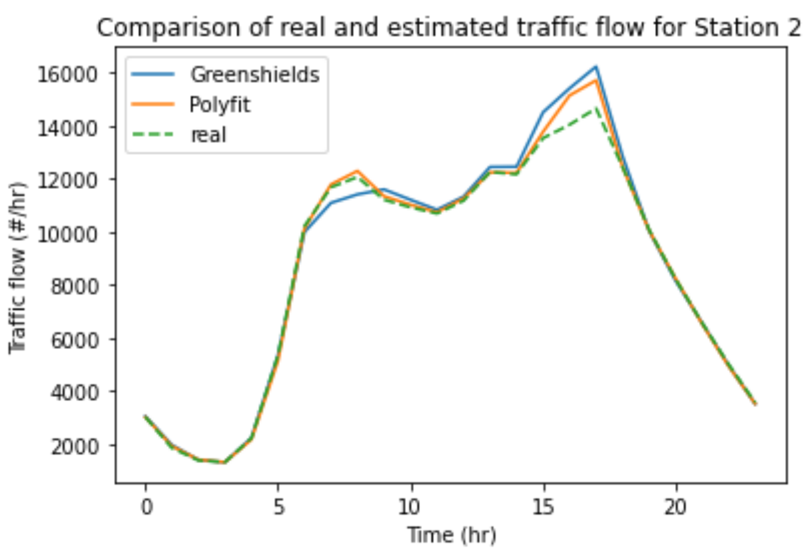
In [5]:
```
# Concatenate traffic density and speed data of Station 1 and 4
density_S1S4 = np.zeros(2*flow_data_size_val2)
speed_S1S4 = np.zeros(2*flow_data_size_val2)
density_S1S4 = np.concatenate((density_S1, density_S4))
speed_S1S4 = np.concatenate((np.array(speed_S1), np.array(speed_S4)))

# Polynomial fit (3rd order) the correlation between speed and traffic
polyfit_coeff = np.polyfit(density_S1S4, speed_S1S4, 3)
rho_v_polyfit = np.poly1d(polyfit_coeff)

# Generate fitted speed for Station 3
speed_S3_fitted = np.zeros(flow_data_size_val2)
speed_S3_fitted = rho_v_polyfit(density_S3)

# Plot the correlation between speed and traffic density for four stati
plt.scatter(density_S1*1.60934, np.array(speed_S1)*1.60934, label='Stat
plt.scatter(density_S2*1.60934, np.array(speed_S2)*1.60934, label='Stat
plt.scatter(density_S3*1.60934, np.array(speed_S3)*1.60934, label='Stat
plt.scatter(density_S4*1.60934, np.array(speed_S4)*1.60934, label='Stat
plt.scatter(density_S3*1.60934, speed_max*1.60934*(1 - np.array(density
plt.scatter(density_S3*1.60934, speed_S3_fitted*1.60934, facecolors='no
plt.title('Correlation between speed and traffic density')
plt.xlabel('Traffic density (#/km)')
plt.ylabel('Speed (km/h)')
plt.legend(loc= 'lower left')
plt.show()
```

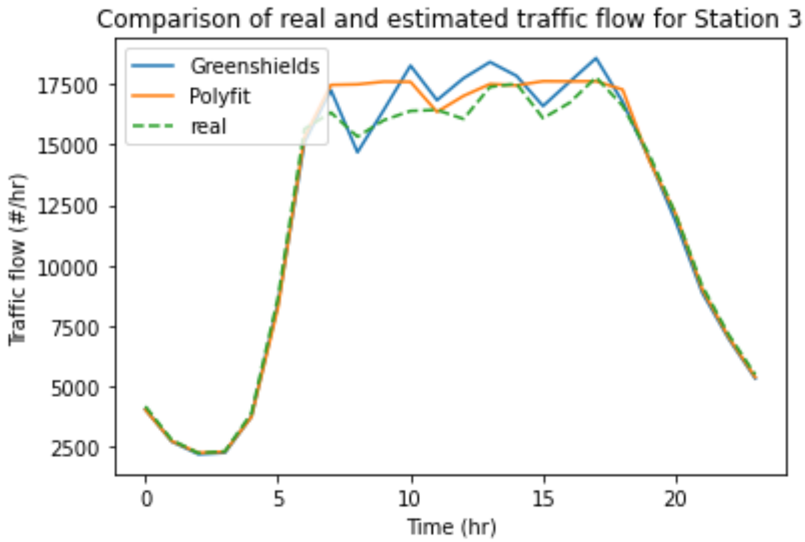


The results show that the speed derived from the Greenshields model (purple square) is generally lower than the actual value (green dot), while the speed fitted from the polynomial regression (black square) approximates the true value well and generally fall within the main sequence formed by the traffic data of four models. The two fitting approaches are utilized to generate the estimated traffic flow for Station 2 and 3 and are plotted in comparison with the true values as shown below.

In [6]:

```
# Calculate estimated traffic flow for Station 2 and 3
volume_S2_estimated_g = np.zeros(flow_data_size_val2)
volume_S3_estimated_g = np.zeros(flow_data_size_val2)
volume_S2_estimated_p = np.zeros(flow_data_size_val2)
volume_S3_estimated_p = np.zeros(flow_data_size_val2)

for i in range(flow_data_size_val2):
    volume_S2_estimated_g[i] = density_S2[i] * speed_max[i] * (1 - dens
    volume_S3_estimated_g[i] = density_S3[i] * speed_max[i] * (1 - dens
    volume_S2_estimated_p[i] = density_S2[i] * rho_v_polyfit(density_S2
    volume_S3_estimated_p[i] = density_S3[i] * rho_v_polyfit(density_S3

# Plot the real and estimated traffic flow for Station 2
plt.plot(time_val2, volume_S2_estimated_g, label='Greenshields')
plt.plot(time_val2, volume_S2_estimated_p, label='Polyfit')
plt.plot(time_val2, volume_S2, '--', label='real')
plt.title('Comparison of real and estimated traffic flow for Station 2'
plt.xlabel('Time (hr)')
plt.ylabel('Traffic flow (#/hr)')
plt.legend(loc= 'upper left')
plt.show()

# Plot the real and estimated traffic flow for Station 3
plt.plot(time_val2, volume_S3_estimated_g, label='Greenshields')
plt.plot(time_val2, volume_S3_estimated_p, label='Polyfit')
plt.plot(time_val2, volume_S3, '--', label='real')
plt.title('Comparison of real and estimated traffic flow for Station 3'
plt.xlabel('Time (hr)')
plt.ylabel('Traffic flow (#/hr)')
plt.legend(loc= 'upper left')
plt.show()
```





The figures show that for both Station 2 and 3 the traffic flow fitted with polynomial regression (orange curve) approximates the true values (green dashed line) better compared with the fitting from the Greenshields model (blue curve).

3.1.4 Iterative approach: Runge-Kutta 4th order (RK4)

In this section, we explore the iterative method to generate the numerical solution for the continuous traffic flow PDE. Since the discretization of PDE equation requires the parametric formulation of variables, we utilize Greenshields model to represent the correlation between speed and traffic density (Eq. 9) although it performs not as good as the polynomial regression in fitting the traffic flow. With this assumption, the Eq. 7 can be further discretized as

$$\frac{\partial \rho}{\partial t} + v_{max}(1 - \frac{2\rho}{\rho_{max}})\frac{\partial \rho}{\partial x} = 0 \qquad \text{(Eq. 11)}$$

Denote $C(\rho) = v_{max}(1 - \frac{2\rho}{\rho_{max}})$ and apply the *finite difference method*, Eq. 11 can be written as

$$\frac{\rho_j^{n+1} - \rho_j^n}{\Delta t} + C(\rho_j^n)\frac{\rho_j^n - \rho_{j-1}^n}{\Delta x} = 0 \qquad \text{(Eq. 12)}$$

Rearrange the time step $n + 1$ term to the left hand side and time step n terms to the right hand side. The final numerical scheme is

$$\rho_j^{n+1} = \rho_j^n + C(\rho_j^n)\frac{\Delta t}{\Delta x}(\rho_j^n - \rho_{j-1}^n) \qquad \text{(Eq. 13)}$$

Haversine formula

In Eq. 13, Δx denotes the distance between two adjacent stations. Given this distance is the distance between two points on Earth (a sphere), the great-circle distance can be calculated via the haversine formula with the latitudes and longitudes of the two points.

$$hav(\Theta) = sin^2(\frac{\phi_2-\phi_1}{2}) + cos(\phi_1)cos(\phi_2)sin^2(\frac{\lambda_2-\lambda_1}{2}) \qquad \text{(Eq. 14)}$$
$$d = r * \Theta = 2r * arcsin(\sqrt{hav(\Theta)}) \qquad \text{(Eq. 15)}$$

where $hav()$ is the haversine funciton, Θ is the central angle, ϕ and λ are latitude and longitude of points, respectively, and r is the Earth radius.

The haversine function is implemented as follows.

In [7]:
```
# Define haversine formula function to calculate the distance on a sphe
def haversine_distance(point_one, point_two):
    lat_1, long_1 = point_one
    lat_2, long_2 = point_two
    earth_radius = 6371  # unit of km
    lat_1_rad = math.radians(lat_1)
    lat_2_rad = math.radians(lat_2)
    delta_lat = math.radians(lat_2 - lat_1)
    delta_long = math.radians(long_2 - long_1)
    hav_theta = math.sin(delta_lat/2)**2 + math.cos(lat_1_rad)*math.cos
    #theta = 2 * math.atan2(math.sqrt(hav_theta), math.sqrt(1-hav_theta
    theta = 2 * math.asin(math.sqrt(hav_theta))
    distance = earth_radius * theta * 0.539957  # 0.539957 is conversic
    return distance
```

Runge-Kutta 4th order

Runge-Kutta 4th order is a commonly used iterative method (normally referred as "RK4") to discretize the ordinary differential equation for generating approximate solutions. This method is internally composed of the Euler method. Given an initial value problem of

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0 \qquad \text{(Eq. 16)}$$

The RK4 method can be expressed as

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \qquad \text{(Eq. 17)}$$
$$t_{n+1} = t_n + h \qquad \text{(Eq. 18)}$$

where h is the time step size, and k_1, k_2, k_3, k_4 are defined as

$$k_1 = f(t_n, y_n) \qquad \text{(Eq. 19)}$$
$$k_2 = f(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}) \qquad \text{(Eq. 20)}$$
$$k_3 = f(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}) \qquad \text{(Eq. 21)}$$
$$k_4 = f(t_n + h, y_n + hk_3) \qquad \text{(Eq. 22)}$$

To apply RK4 in Eq. 13, the $f()$ in RK4 method formulates the differential equation of continous traffic flow, specifically $f()$ represents the derivative of traffic density with respect to time ($\frac{\partial \rho}{\partial t}$). The implementation of RK4 in simulating the traffic flow is shown below with the estimated results for Station 2 and 3 compared with the true values and the fitted values from the analytical derivation.

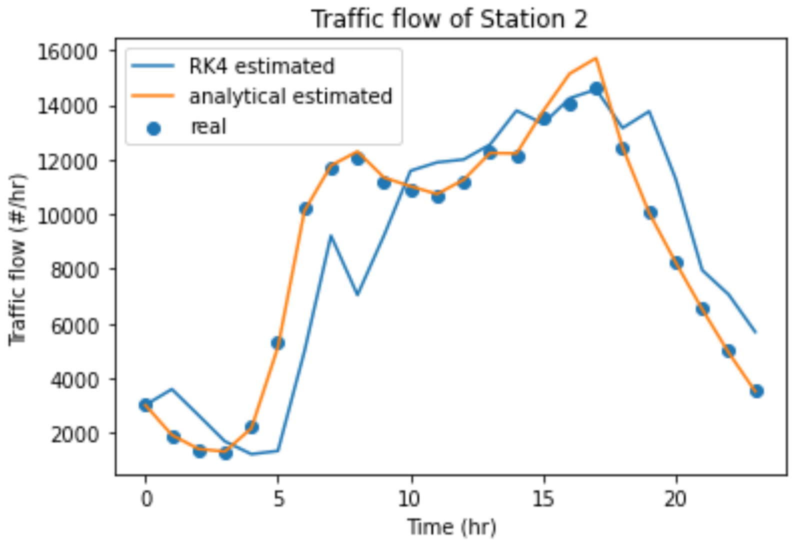
In [8]:

```
# Define traffic flow model ODE (time derivative of traffic density)
def drho_dt(rho, v_cur, v_next, distance):
    return -rho * ((v_next-v_cur)/distance)  # Note this ODE does not i

# Define RK4 function
def rungeKutta4th(drho_dt, v_cur, v_next, distance, t0, rho0, h, n):

    # Initialize time and traffic density array
    t_array = np.zeros(n+1)
    rho_array = np.zeros(n+1)

    # Assign initial conditions
    t_array[0] = t0
    rho_array[0] = rho0

    # Interatively calculate traffic density of each time step
    for i in range(1,n+1):

        # Calculate k1 to k4 values in RK4 formula
        k1 = drho_dt(rho_array[i-1], v_cur, v_next, distance)
        k2 = drho_dt(rho_array[i-1] + 0.5*k1, v_cur, v_next, distance)
        k3 = drho_dt(rho_array[i-1] + 0.5*k2, v_cur, v_next, distance)
        k4 = drho_dt(rho_array[i-1] + k3, v_cur, v_next, distance)

        # Assign updated t and calculated rho to the corresponding time
        t_array[i] = t_array[i-1] + h
        rho_array[i] = rho_array[i-1] + (1.0/6.0)*h*(k1 + 2*k2 + 2*k3 +

    # Return iteratively derived traffic density at time step n
    return rho_array[n]
```

In [9]:

```
# Define input parameters for RK4
t_initial = 0
num_iteration = 1
sub_step_size = 0.01

# Define station coordinates
coord_S1 = [33.82764, -84.34427]
coord_S2 = [33.88379, -84.26857]

# Assign time variable for Station 1 and 2
time_S1 = time_val2
time_S2 = time_val2
```

In [10]:
```
# Convert speed array to Numpy array
speed_S1_array = np.array(speed_S1)
speed_S2_array = np.array(speed_S2)

# Initialize empty array to store results
rho_S1_initial = np.zeros(len(time_S1))
rho_S2 = np.zeros(len(time_S1))
rho_S2_approx = np.zeros(len(time_S1))
best_step_size = np.zeros(len(time_S1))
volume_S2_approx = np.zeros(len(time_S1))

# Assign values for the first row of density arrays for Station 2
rho_S2[0] = density_S2[0]
rho_S2_approx[0] = rho_S2[0]  # the initial traffic density is assigned
volume_S2_approx[0] = rho_S2[0] * speed_S2[0]

# Iteratively calculate the estimated traffic density of all time steps
for i in range(0, len(time_S1)-1):

    # Assign speed values of current time step and next time step for S
    v_cur = speed_S1[i]
    v_next = speed_S1[i+1]

    # Calculate distance between Station 1 and 2
    dist = haversine_distance(coord_S1, coord_S2)

    # Assign initial traffic density value of all time steps for Statio
    rho_S1_initial[i] = volume_S1[i] / speed_S1[i]

    # Create an array with various time step size
    step_size = np.arange(sub_step_size, 1, sub_step_size)

    # Collect all results of traffic density of Station 2 of next time 
    rho_S2_step_size_res = np.zeros(len(step_size))
    for j in range(0, len(step_size)):
        rho_S2_step_size_res[j] = rungeKutta4th(drho_dt, v_cur, v_next,

    # Select the simulated traffic density value of Station 2 that appr
    rho_S2[i+1] = density_S2[i+1]
    rho_S2_approx[i+1] = min(rho_S2_step_size_res, key=lambda x: abs(x-

    # Estimate the traffic flow value of Station 2 for the next time st
    volume_S2_approx[i+1] = rho_S2_approx[i+1] * speed_S2[i+1]

# Plot RK4, analytical estimated traffic flow of Station 2 and 3 compar
plt.scatter(time_S2, volume_S2, label='real')
plt.plot(time_S2, volume_S2_approx, label='RK4 estimated')
plt.plot(time_S2, volume_S2_estimated_p, label='analytical estimated')
plt.title('Traffic flow of Station 2')
plt.xlabel('Time (hr)')
plt.ylabel('Traffic flow (#/hr)')
plt.legend(loc= 'upper left')
plt.show()
```



The above figure shows that RK4 estimated results have relatively large discrepancy from the ground true values compared with the analytical derivation. To be noted, this RK4 implementation simplifies the PDE as the spatially homogeneous traffic density. The consideration of spatially heterogeneous traffic density scenario is still in progress.

3.2 Complex model [in progress]

Given many real-world factors in traffic are not considered in the basic model, we expect the performance of the simple model to simulate the traffic data is poor. As a result, our second model would be a non-linear model that incorporates several effects, such as traffic light, intersection, etc. We haven't figured out what type of non-linear model would be best to fit in this section. We will work on the development of the complex model after literature review and further group discussions. Same as the first model, after the implementation, the validity of the model would be tested on the real traffic data to see if the fitting performance is improved. It is expected that parameter tuning would be involved.

Reference

[1] Haberman, Richard. Mathematical models: mechanical vibrations, population dynamics, and traffic flow. Vol. 21. Siam, 1998.
[2] http://www.norbertwiener.umd.edu/Education/m3cdocs/Presentation2.pdf (http://www.norbertwiener.umd.edu/Education/m3cdocs/Presentation2.pdf)
[3] H. Holmes. Introduction to the Foundations of Applied Mathematics. Springer (2009)
[4] https://www.math.nyu.edu/faculty/childres/traffic2.pdf (https://www.math.nyu.edu/faculty/childres/traffic2.pdf)
[5] https://www.sciencedirect.com/science/article/pii/S0377042706002159#bib8 (https://www.sciencedirect.com/science/article/pii/S0377042706002159#bib8)