CS 3430: S19: SciComp with Py
Assignment 9
Estimating Bee Traffic Levels with Integral Approximation

Vladimir Kulyukin
Department of Computer Science
Utah State University

March 8, 2019

Learning Objectives

1. Definite Integral Approximation
2. Data Analysis
3. CSV File Parsing

Introduction

In this assignment, we'll build up on the previous assignment to do some real data analysis with definite integral approximation. The folder `bee_traffic_estimates` contains 292 CSV files. Each file is automatically generated by BeePi, my electronic beehive monitoring system, from 30-second mp4 videos of hive landing pads. Those of you who attended Lecture 16 should recall the BeePi video demo I showed in class. Each file name starts with the unique id of a monitored hive (e.g., `192_168_4_5`) followed by a timestamp of when the video was taken (e.g., `2018-07-01_08-00-10`, which means that this video was taken on July 1, 2018 at 8:00:10 a.m.). The timestamps use the 24-hour format.

If viewed as a table, each CSV file has 25 rows and 4 colums. As shown in the following table.

TIME (IN SECS)	UPWARD	DOWNWARD	LATERAL
5	0.4	0.6285714286	0.4285714286
6	0.2285714286	0.6	0.4571428571
...	...	...	...
28	0.7936507937	2.3015873016	0.873015873

The semantics of each row is as follows. The first column is the number of seconds into the video; the second column is the estimate of how many bees left the beehive (i.e., moved upward); the third column is the estimate of how many bees came into the beehive (i.e., moved downward); the fourth column is the estimate of how many bees moved laterally on or in front of the landing pad. For example, the last row in the table above says that at the 28th second of the video approximately 0.79 bees moved upward, 2.3 bees moved downward, and 0.87 bees moved laterally.

Parsing CSV files in Python is straightforward. Let's write a function that displays a CSV file row by row.

```
import csv
def display_csv_file(csv_file_path):
  with open(csv_file_path, 'r') as instream:
    reader = csv.reader(instream, delimiter = ",")
    for row in reader:
      print(row)
```

Let's test it in the Py shell.

```
>>> display_csv_file('192_168_4_5-2018-07-01_08-00-10.csv')
['TIME IN SECS', 'UPWARD', 'DOWNWARD', 'LATERAL']
['5', '0.4', '0.628571428571', '0.428571428571']
['6', '0.228571428571', '0.6', '0.457142857143']
['7', '0.371428571429', '1.42857142857', '0.6']
['8', '0.571428571429', '1.25714285714', '1.28571428571']
['9', '0.342857142857', '2.54285714286', '0.228571428571']
['10', '0.2', '2.77142857143', '0.0857142857143']
['11', '0.485714285714', '1.14285714286', '0.257142857143']
['12', '0.914285714286', '1.28571428571', '0.8']
['13', '0.857142857143', '2.34285714286', '1.08571428571']
['14', '0.685714285714', '1.51428571429', '0.285714285714']
['15', '0.485714285714', '2.14285714286', '0.942857142857']
['16', '0.685714285714', '1.37142857143', '0.8']
['17', '1.17142857143', '1.2', '1.74285714286']
['18', '0.714285714286', '2.11428571429', '0.342857142857']
['19', '0.8', '0.8', '0.285714285714']
['20', '1.14285714286', '0.942857142857', '0.342857142857']
['21', '0.714285714286', '2.05714285714', '0.885714285714']
['22', '1.02857142857', '0.685714285714', '0.714285714286']
['23', '0.771428571429', '0.828571428571', '0.4']
['24', '0.771428571429', '1.74285714286', '0.914285714286']
['25', '0.285714285714', '1.8', '0.314285714286']
['26', '0.828571428571', '1.48571428571', '0.685714285714']
['27', '0.914285714286', '0.942857142857', '0.371428571429']
['28', '0.793650793651', '2.30158730159', '0.873015873016']
```

Let's write a function that turns a CSV file into a lookup dictionary where the keys are seconds and the values are 3-tuples with the upward, downward, and lateral estimates.

```
def read_csv_file(csv_file_path):
  fd = {}
  with open(csv_file_path, 'r') as instream:
    reader = csv.reader(instream, delimiter = ",")
    reader.next() #to skip the header (i.e., column names)
    for row in reader:
      secs, up, down, lat = int(row[0]), float(row[1]), \
                            float(row[2]), float(row[3])
      fd[secs] = (up, down, lat)
    return fd
```

Let's test this function in the shell.

```
>>> fd = read_csv_file('192_168_4_5-2018-07-01_08-00-10.csv')
>>> fd[5]
(0.4, 0.628571428571, 0.428571428571)
>>> fd[28]
(0.793650793651, 2.30158730159, 0.873015873016)
```

Problem 1: Plotting Bee Traffic (0.5 points)

The first tool that we'll build is the the function `plot_bee_traffic(fp)` that takes a path to a CSV file `fp`, loads it into a dictionary with `read_csv_file`, and plots the time-dependent estimates of the upward moving bees as a red line, of the downward moving bees as a green line, and of the latterly moving bees as a blue line, as shown in Fig. 1.

Save your code in `hw09_s19.py`.

Problem 2: Estimating Bee Traffic Levels with Integral Approximation (1.5 points)

We can estimate the levels of upward, downward, and lateral bee traffic in a video by computing the areas under the corresponding curves, i.e., approximating definite integrals. We'll confine ourselves to Simpson's rule in this assignment. Implement the function `sp_approx(f, a, b, n)` that takes a Python function `f`, the upper and lower limits of an interval, `a` and `b`, and a number of subintervals `n` in the partition. Note that the last three parameters are floats. Since we are not doing any differentiation or antidifferentiation, there is no need for us to use objects such as `var`, `const`, `pwr`, etc.

I used the following formula to implement Simpson's rule.

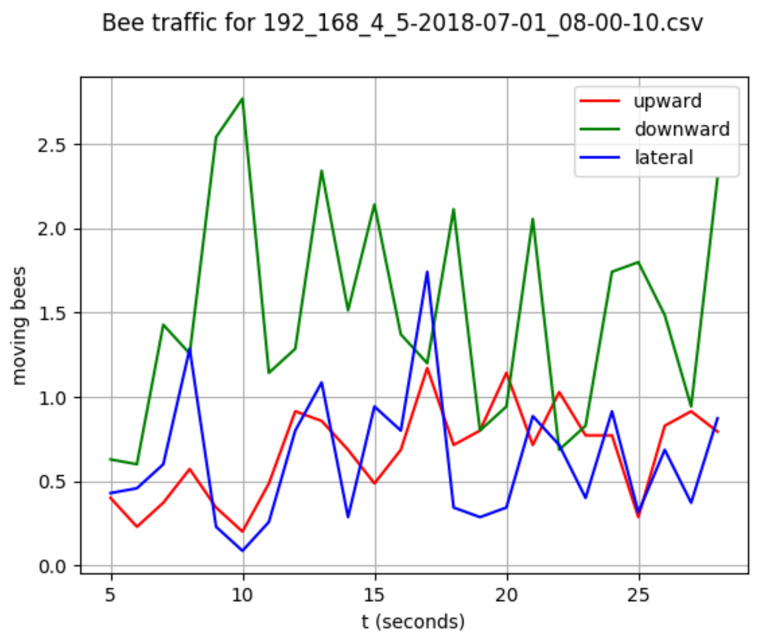

Figure 1: Bee traffic plots.

$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + ... + 4f(x_{n-1}) + f(x_n)),$$

where $\Delta x = (b - a)/n$ and $x_i = a + i\Delta x$.

Save your code in `hw09_s19.py`.

Let's run a few tests with `sr_approx` in the Shell.

```
>>> sr_approx(lambda x: x**2, 0, 2, 10)
2.6666666666666665
>>> sr_approx(lambda x: x**3, 1, 5, 100)
156.00000000000006
```

We can check how good our approximation is by comparing the above results with the implementations of the quadrature and simpson approximations from `scipy.integrate`.

```
>>> x = np.linspace(0, 2, 10)
>>> y = np.array([i**2 for i in x])
>>> scipy.integrate.simps(y, x)
2.6684956561499775
>>> scipy.integrate.quad(lambda x: x**2, 0, 2)[0]
2.666666666666667
>>> x = np.linspace(1, 5, 100)
>>> y = np.array([i**3 for i in x])
>>> scipy.integrate.simps(y, x)
156.00009893857461
>>> scipy.integrate.quad(lambda x: x**3, 1, 5)[0]
156.0
```

Looks like we are pretty close. Of course, your results may be slightly different depending on the rounding error on your system. But, they should be in the same ballpark.

Let's implement some machinery that would allow us to take a time value, a dictionary constructed by `read_csv_file`, and a movement direction (`'u'` – for upward, `'d'` – for downward, and `'l'` – for lateral) and return the appropriate estimate of the bee traffic level at that time and in the specified direction.

```
def bee_traffic_estimate(t, md='u', fd={}):
  assert md == 'u' or md == 'd' or md == 'l'
  vals = fd.get(int(t))
  if vals is None:
    return None
  elif md == 'u':
    return vals[0]
  elif md == 'd':
    return vals[1]
  elif md == 'l':
    return vals[2]

def make_bee_traffic_estimator(fd, md):
  assert md == 'u' or md == 'd' or md == 'l'
  return lambda t: bee_traffic_estimate(t, md=md, fd=fd)
```

With this machinery in place, we can estimate the three directional levels of bee traffic at a given time in a given CSV file.

```
def test(csv_fp):
  FD = read_csv_file(csv_fp)
  up_bte = make_bee_traffic_estimator(FD, 'u')
  down_bte = make_bee_traffic_estimator(FD, 'd')
  lat_bte = make_bee_traffic_estimator(FD, 'l')
  print(sr_approx(up_bte, 5, 28, 23))
  print(sr_approx(down_bte, 5, 28, 23))
  print(sr_approx(lat_bte, 5, 28, 23))
```

```
>>> test('192_168_4_5-2018-07-01_08-00-10.csv')
15.5597883598
33.491005291
14.1291005291
>>> test('192_168_4_5-2018-07-01_08-30-10.csv')
14.5726190476
31.3642857143
12.8869047619
>>> test('192_168_4_5-2018-07-01_16-30-10.csv')
40.4714285714
56.0285714285
27.3571428571
```

Implement the function `bee_traffic_stats(fd)` that takes a dictionary returned by `read_csv_file` and returns three directional bee traffic level approximations by using `sr_approx` to integrate the appropriate functions over the interval $[5, 28]$ with a partition of 23 subintervals. In other words, this function returns a 3-tuple where the first element is the upward traffic level estimate, the second element is the downward traffic level estimate, and the third element is the lateral traffic level estimate. Note that the values $[5, 28]$ and 23 are the same for every CSV file. You can and should `make_bee_traffic_estimator` in your definition.

Save your code in `hw09_s19.py`.

Let's run a few tests.

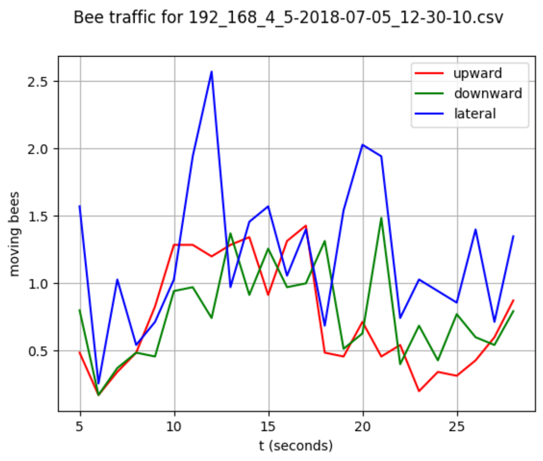

Figure 2: Bee traffic plots of the csv file with the smallest gap.

```
>>> fd = read_csv_file('192_168_4_5-2018-07-02_16-30-10.csv')
>>> bee_traffic_stats(fd)
(40.47142857141933, 56.028571428527336, 27.357142857131997)
>>> fd = read_csv_file('192_168_4_5-2018-07-01_14-00-10.csv')
>>> bee_traffic_stats(fd)
(38.048677248690325, 36.675132275122, 27.971428571415665)
>>> fd = read_csv_file('192_168_4_5-2018-07-01_18-00-10.csv')
>>> bee_traffic_stats(fd)
(24.505820105825652, 22.737566137565995, 24.006349206362394)
```

Problem 3: Analyzing Bee Traffic Levels (3 points)

Let's implement several functions to analyze bee traffic levels.

Problem 3.1

Implement the function `find_smallest_up_down_gap_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the smallest gap between the upward and downward bee traffic estimates. The gap is measured as the absolute value of the difference between the upward and downward estimates. This function returns a 5-tuple where the first element is the path to the csv file with the smallest gap, the second element is the upward estimate, the third element is the downward estimate, the fourth element is the lateral estimate, and the fifth element is the gap.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 2 shows the plot displayed by the test.

```
def test_smallest_up_down_gap(csv_dir):
  fp, u, d, l, gap = find_smallest_up_down_gap_file(csv_dir)
  print(fp, u, d, l, gap)
  plot_bee_traffic(fp)

>>> test_smallest_up_down_gap('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-05_12-30-10.csv',
 16.948148148144668, 16.950264550266994, 27.06878306878, 0.002116402122325667)
```

Implement the function `find_largest_up_down_gap_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the largest gap between the upward and downward estimates. The gap is measured as the absolute value of the difference between the upward and downward estimates. This function returns a 5-tuple where the first element is the path to the csv file with the smallest gap, the second element is the upward estimate, the third element is the downward estimate, the fourth element is the lateral estimate, and the fifth element is the gap.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 3 shows the plot displayed by the test.

```
def test_largest_up_down_gap(csv_dir):
  fp, u, d, l, gap = find_largest_up_down_gap_file(csv_dir)
  print(fp, u, d, l, gap)
  plot_bee_traffic(fp)

>>> test_largest_up_down_gap('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-03_09-30-10.csv',
 16.543915343913326, 49.33439153437398, 14.739682539683532, 32.79047619046065)
```

Problem 3.2

Implement the function `find_max_up_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the largest upward traffic estimate. This function returns a 4-tuple where the first element is the path to the csv file with the largest upward traffic estimate, the second element is the upward estimate, the third element is the downward estimate, and the fourth element is the lateral estimate.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 4 shows the plot displayed by the test.

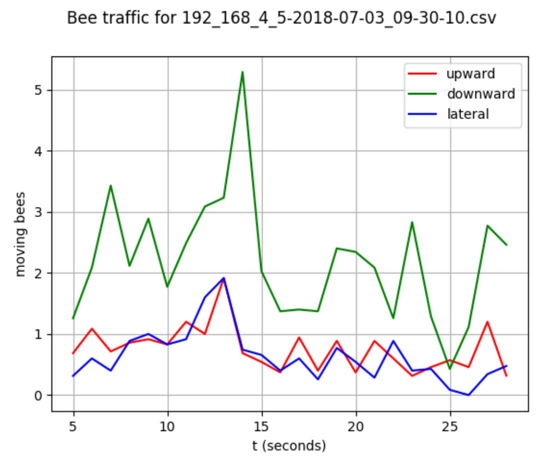

Figure 3: Bee traffic plots of the csv file with the largest gap.

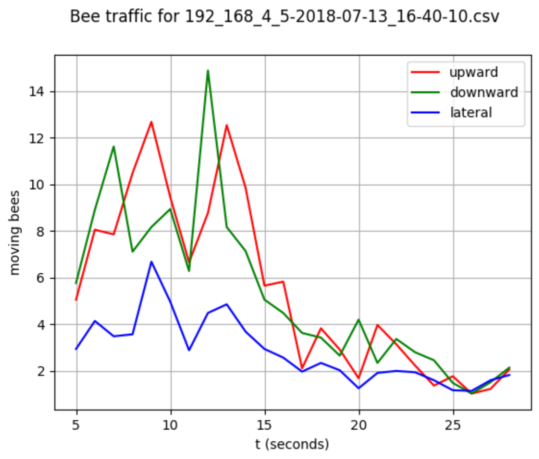

Figure 4: Bee traffic plots of the csv file with the largest upward estimate.

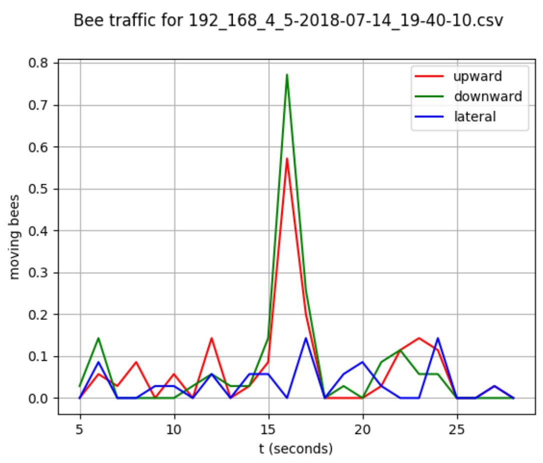

Figure 5: Bee traffic plots of the csv file with the smallest upward estimate.

```
def test_max_up(csv_dir):
  fp, u, d, l = find_max_up_file(csv_dir)
  print(fp, u, d, l)
  plot_bee_traffic(fp)

>>> test_max_up('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-13_16-40-10.csv',
 126.81164021162336, 126.42857142857669, 64.94179894179999)
```

Implement the function `find_min_up_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the smallest upward traffic estimate. This function returns a 4-tuple where the first element is the path to the csv file with the smallest upward traffic estimate, the second element is the upward estimate, the third element is the downward estimate, and the fourth element is the lateral estimate.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 5 shows the plot displayed by the test.

```
def test_min_up(csv_dir):
  fp, u, d, l = find_min_up_file(csv_dir)
  print(fp, u, d, l)
  plot_bee_traffic(fp)

>>> test_min_up('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-14_19-40-10.csv',
 1.9047619047630002, 1.990476190477, 0.8380952380950668)
```

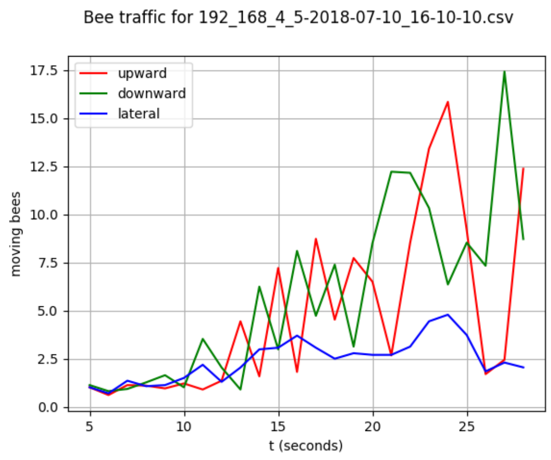

Figure 6: Bee traffic plots of the csv file with the largest downward estimate.

Problem 3.3

Implement the function `find_max_down_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the largest downward bee traffic estimate. This function returns a 4-tuple where the first element is the path to the csv file with the largest upward traffic estimate, the second element is the upward estimate, the third element is the downward estimate, and the fourth element is the lateral estimate.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 6 shows the plot displayed by the test.

```
def test_max_down(csv_dir):
  fp, u, d, l = find_max_down_file(csv_dir)
  print(fp, u, d, l)
  plot_bee_traffic(fp)

>>> test_max_down('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-10_16-10-10.csv',
 103.70793650793468, 129.4433862434233, 55.50687830690132)
```

Implement the function `find_min_down_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the smallest downward bee traffic estimate. This function returns a 4-tuple where the first element is the path to the csv file with the largest upward traffic estimate, the second element is the upward estimate, the third element is the downward

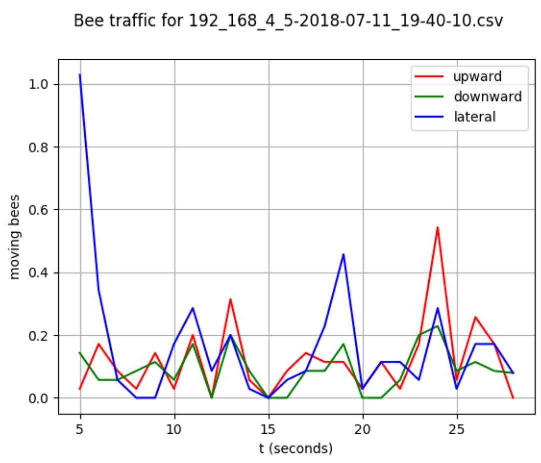

Figure 7: Bee traffic plots of the csv file with the smallest downward estimate.

estimate, and the fourth element is the lateral estimate.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 7 shows the plot displayed by the test.

```
def test_min_down(csv_dir):
  fp, u, d, l = find_min_down_file(csv_dir)
  print(fp, u, d, l)
  plot_bee_traffic(fp)

>>> test_min_down('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-11_19-40-10.csv',
 2.8095238095256665, 1.8835978835987002, 3.3597883597886993)
```

Problem 3.4

Implement the function `find_max_lat_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the largest lateral bee traffic estimate. This function returns a 4-tuple where the first element is the path to the csv file with the largest upward traffic estimate, the second element is the upward estimate, the third element is the downward estimate, and the fourth element is the lateral estimate.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 8 shows the plot displayed by the test.

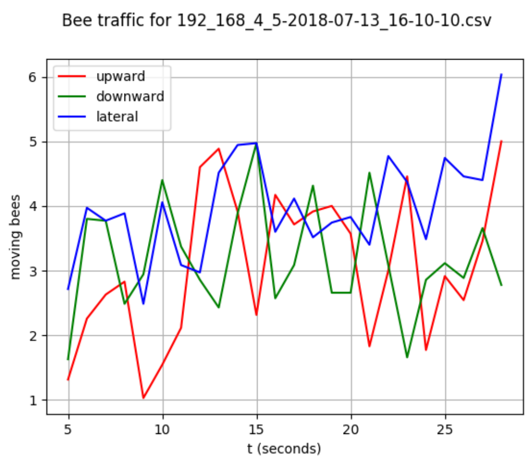

Figure 8: Bee traffic plots of the csv file with the largest lateral estimate.

```
def test_max_lat(csv_dir):
  fp, u, d, l = find_max_lat_file(csv_dir)
  print(fp, u, d, l)
  plot_bee_traffic(fp)

>>> test_max_lat('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-13_16-10-10.csv',
 69.81904761906999, 73.31640211637666, 89.96296296296669)
```

Implement the function `find_min_lat_file(csv_dir)` that takes a directory of csv files returned by the BeePi vision system and finds a file with the smallest lateral bee traffic estimate. This function returns a 4-tuple where the first element is the path to the csv file with the largest upward traffic estimate, the second element is the upward estimate, the third element is the downward estimate, and the fourth element is the lateral estimate.

Save your code in `hw09_s19.py`.

Let's run a test. Fig. 9 shows the plot displayed by the test.

```
def test_min_lat(csv_dir):
  fp, u, d, l = find_min_lat_file(csv_dir)
  plot_bee_traffic(fp)
  print(fp, u, d, l)

>>> test_min_lat('bee_traffic_estimates/')
('bee_traffic_estimates/192_168_4_5-2018-07-14_19-40-10.csv',
 1.9047619047630002, 1.990476190477, 0.8380952380950668)
```

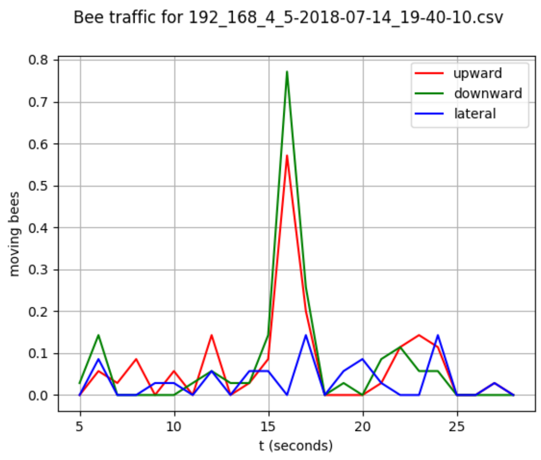

Figure 9: Bee traffic plots of the csv file with the smallest lateral estimate.

What to Submit

You have to submit `hw09_s19.py`. You should also submit all the files needed to run your code. Zip your entire working directory in `hw09.zip` and submit it via Canvas.

And here comes my perpetual refrain! Do not change the names of the files that were given to you or the names of the functions you are asked to implement. The unit tests that I write for the grader every week depend on these names remaining the same.

Happy Hacking!