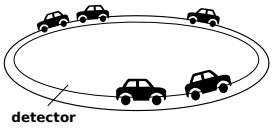
Traffic flow

1 Introduction

In this assignment you will be modelling the flow of car traffic and the formation of traffic jams. This is another classic topic in mathematical modelling, about which much has been written in the literature. Having good mathematical models is of obvious use in the design of new roads and in traffic light planning, as it can help to avoid slow traffic and also lead to decrease of fuel use.

In the simplest models for traffic flow, we consider a stretch of road of length L, and impose periodic boundary conditions (if a car leaves the end of the stretch, it appears again at the beginning with the same speed). This corresponds to cars driving on a circular road, which is obviously not very realistic, but it makes modelling a lot easier. If we put a detector on the road, we can measure how many vehicles pass during a particular time interval ΔT. If ΔN vehicles pass the detector in a time ΔT, then the *flow* q is



Cars on a road with periodic boundary conditions (circular road) and a detector to measure the flow. Cars drive in one direction only, and cannot overtake.

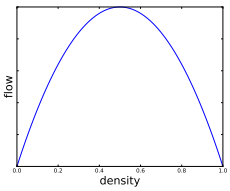
$$q = \frac{\Delta N}{\Delta T} \qquad \text{(fixed position)}\,. \tag{1.1}$$

As indicated, this is a measurement 'at fixed position'. We can alternatively take a snapshot of the entire stretch, and count how many cars there are. The *density* of cars ρ is given by the number of cars per unit road length at a fixed time, that is

$$\rho = \frac{N}{L} \qquad \text{(fixed time)}\,. \tag{1.2}$$

In contrast to the flow, this is a measurement performed at a fixed instance of time. These two quantities are the most important in traffic flow, and one often looks at graphs which show the relation between these two, the so-called *fundamental diagram* of traffic flow.

Different models produce different fundamental diagrams, but they all have to produce some of the qualitative features of the diagram on the right. That is, for low density (few cars per kilometer), we expect that the flow increases with the density. For very large density, however, when the cars sit bumper-to-bumper, we expect the flow to be small again: in a traffic jam, there are many cars, but very few drive past the detector because they all move very slowly.



Sample fundamental diagram for a traffic model.

One of the key things which traffic modelling tries to explain is the formation of spontaneous traffic jams. These are strong peaks of traffic density, which appear 'out of the blue', without there being a clear cause (such as an accident or traffic light turning red). Another feature which we would like our models to reproduce and explain is the fact that real-world fundamental diagrams (obtained by measuring ρ and q on actual roads) often do not look as smooth as in 1, but rather exhibit a sudden change at a particular traffic density.

In the practicals so far, we have been using different methods to model physical systems. The first is a 'discrete' approach (followed in the first chapter on population dynamics) in which the world is made into a discrete model which is updated iteratively. The second approach is one which uses ordinary differential equations (in the continuous-time approach for population dynamics). In the present assignment we will see both of these again in two different models for traffic flow.

Question 1:

If a car driving at constant speed v jams on the brakes and decelerates at the maximal rate $a_{\max}$, how long does it need to stop completely? We will call that braking time Δt.

Notice that a_{max} is a negative acceleration and the equation describing the movement of the car is

$$\ddot{x} = -a_{\text{max}} \,. \tag{1.3}$$

What distance Δx does the car travel before it has stopped completely (expressed in terms of v and a_{max})?

Now assume that the cars all drive at the same constant speed v_0, and that they are separated by the distance Δx you found above. That is, if the car in front comes to a stop instantaneously, the following car can just manage to avoid a collision. The total space for each car is thus $\Delta L = \Delta x + l_c$ where l_c is the length of each car. Compute the flow q past a fixed detector in terms of v_0, a_{max} and l_c.

Set the maximal acceleration equal to $a_{\text{max}} = 5\text{m/s}^2$ (about half the gravitational acceleration) and the car length to $l_c = 5\text{m}$, then plot the flow q as a function of v_0.

Is this first model any good, or is it too simple? To answer that question we will build two traffic models which take into account more of the dynamics of cars.

2 Discrete models

Our first improved model for traffic flow will be a discrete one, in which the road is split up into segments, each of which can be either empty or contain a single car. It is a common practice to make the road circular, so that cars never leave the road. Each segment has a length equal to the length of a car plus the minimal space between cars; we will take it to be about 7.5m. The position of the n-th car is denoted by x_n. Each car can have a different velocity v_n, which is given in terms of the number of segments per second. An example is drawn below.

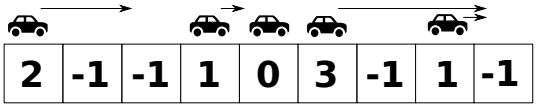


Figure 1.1: A road discretised in segments. The number indicates the velocity. For example, the third car has zero velocity. The fourth car has velocity 3 segments/second which corresponds to 81 km/h. If it does not brake (we will see how we model this shortly) it would crash into the last car. Segments without a car are labelled with $v = -1$ (we do not allow cars to drive backwards, so we can safely use this value).

At every time step, we now update the car velocities and positions according to the following rules:

1. Each car accelerates as long as it has not reached the maximal legal speed v_{max} yet, $v_n \to \min(v_n + 1, v_{\text{max}})$.
2. If a car gets too close to the next car, it decelerates, $v_n \to \min(v_n, h_n - 1)$. Here $h_n = x_{n+1} - x_n$ is the so-called *headway*, the distance to the next car. This, together with rule 4 below, implies that cars will never hit cars in front of them, even if those cars brake to standstill instantaneously.
3. Each car slows down randomly, $v_n \to \max(v_n - 1, 0)$ with probability p_{slowdown}. This models, in a very crude way, random driver behaviour.
4. The car then moves as $x_n \to x_n + v_n$.

These rules do not necessarily capture a lot of realistic car driver behaviour, but they are simple to implement and can easily be extended to more complicated rules.

In the file `traffic_discrete_base.py` you will find a base class `TrafficDiscreteBase` with helper methods to deal with simulation of this discrete system. You will need to implement several additional methods yourself in a subclass.

 Copyright © 2015-2017 Kasper Peeters and Bernard Piette; Durham University.

Coding task 1:

Start by carefully reading the `__init__` method of the `TrafficDiscreteBase` class to become familiar with the class variables you will need to use (note in particular that `self.v` stores the velocities, which were shown for a very simple example in figure 1.1). Also read the various class member functions. You do not need to understand the details of how each function works, but you must understand what they actually do by reading the docstrings. Then do the following:

- Create a file `traffic_discrete.py` in which you create a subclass `TrafficDiscrete` which inherits from `TrafficDiscreteBase`. You will need to import `numpy` and `traffic_discrete_base` at the top of the file.
- Implement the `step(self)` class function. It should implement the four traffic rules described above on the `self.v` array, which holds the velocities for all segments of the road (or -1 if there is no car). It should *also* update the `self.detector` variable, which counts the number of cars that pass the detector at the end of the road.

  To write the function `step(self)` we need to know where each cars is (their index position in the class variable `self.v`) as well as the distance separating them from the car in front (the headway). This can be fiddly to program, but line 5 and 6 in the sample code `npsample.py` (displayed below) show you how to do it quickly, using the sample road set in line 4 (identical to figure 1.1).

  file: `npsample.py`

```
1  import numpy as np
2
3  # Aspects of the sample road in figure 1.1.
4  v       = np.array([2,-1,-1,1,0,3,-1,1,-1])
5  cars    = np.where(v!=-1)[0]
6  headway = np.mod( np.roll(cars, -1) - cars, len(v) )
7
8  print("v          = ", v[cars])
9  print("cars index = ", cars)
10 print("v[cars]    = ", v[cars])
11 print("headway    = ", headway)
12 print()
13
14 # Generate an array with a random number 0-1 for each segment.
15 rnd     = np.random.rand(len(v))
16 print("rnd         = ", rnd)
17
18 # Which of these values are < 0.3 ?
19 rnd_tf  = rnd < 0.3
20 print("rnd_tf      = ", rnd_tf)
21
22 # Select one of two alternatives based on rnd_tf.
23 choices = np.choose( rnd_tf, [ -1, 1 ] )
24 print("choices     = ", choices)
25
26 # Ensure car positions satisfy the periodic boundary conditions.
27 cars   += 3
28 cars    = np.remainder( cars, len(v) )
29 print("cars, moved = ", cars)
30
31 # Clip velocities to a maximum of '3'.
32 v = np.minimum(v, 3)
33 print("v, clipped  = ", v)
```

  Run this code and look at the results displayed on the screen to understand what the arrays `cars`, `headway` and so on correspond to.

  The rest of the coding task can be implemented using explicit loops over the elements of `self.v` (provided you do this correctly it will give you full marks). However, it can be cleaner and more readable if you use numpy methods. The program `npsample.py` illustrate some other numpy methods which you may find useful for this programming task.

Copyright © 2015-2017 Kasper Peeters and Bernard Piette; Durham University.

You of course need to adapt all of this to solve the task at hand. Do not forget to comment on why your code does things.

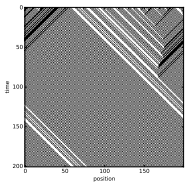
- Implement the `flow_density(self, density, steps)` function, which should return the flow given a `density`, as computed using a simulation running for `steps` iterations. Before you start measuring the flow, let the system evolve for `steps//10` steps to settle down. Make this function return a tuple containing `(density, flow)`.

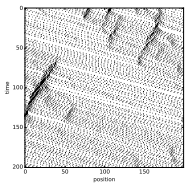
  Make good use of the class functions already defined in `TrafficDiscreteBase`. In particular, use the class function `fill_road_randomly(self)` to initialise the road with cars with an *approximate* density `self.density`.

  This function will only work properly if your `step(self)` function updates the `self.detector` variable.

It is a good idea to regularly test your code while writing it. To do so, you can put the following test block at the bottom of your `traffic_discrete.py` file,

```
54 # Test code; only run when this file is not imported as a module.
55
56 if __name__=="__main__":
57     # Totally deterministic behaviour (which reduces the model to
58     # something known as 'Rule 184').
59     tr=TrafficDiscrete(density=0.5, p_slowdown=0, vmax=1)
60     tr.evolve(200)
61     tr.make_plot()
62
63     # More complicated case with random behaviour.
64     tr=TrafficDiscrete(density=0.2, p_slowdown=0.2, vmax=4)
65     tr.evolve(200)
66     tr.make_plot()
```





It should produce figures similar to the ones displayed in the margin (not identical, because of the random distribution of vehicles and random behaviour of the drivers).

Question 2:

(Answer this question at the end of your essay, just after the references, if you cannot make it fit into the flow of the text).

Why do you need to return not only the flow but also the density when you implement the `flow_density(self, density)` function, even though we already passed `density` as a parameter to this function? Hint: look carefully at how the `fill_road_randomly` function of the base class works.

Question 3:

Run the test code `traffic_discrete_test.py`. It will produce various graphs, detailed below, which you should include in your essay. Comment on the characteristic features of these graphs.

a) Three space-time figures of traffic comparing the effect of different densities of cars. Explain what you see in these plots and explain how you can see the formation of spontaneous traffic jams.

b) A velocity-density plot for purely deterministic traffic, as well as a similar plot where drivers slow down randomly. Comment on the appearance of a 'phase transition' in the purely deterministic case.

c) Two flow-density plots (for various values of $v_{\max}$), one for purely deterministic traffic and one which includes random slowdown. Comment on their differences and their relation to figure 1.

The program saves each figure as a PDF file in the current folder. Be aware that the program can take a few minutes to complete.

Copyright © 2015-2017 Kasper Peeters and Bernard Piette; Durham University.

3 Deterministic ODE model

The discrete traffic model we have analysed above shows some interesting behaviour even when the behaviour of drivers is fully deterministic (i.e. when `p_slowdown=0`), which you may not have expected. In the present section we will analyse a different model, which is completely deterministic (no random behaviour of cars except in their initial position on the road). We will make it a continuous-time model, that is, all car positions are given by functions of a continuous time variable t. We will keep the road one-dimensional (no side roads or overtaking) and again implement periodic boundary conditions (circular road).

The equation of motion for the n-th car is simply given by Newton's law,

$$M\ddot{x}_n(t) = F(t)\,, \tag{1.4}$$

where M is the car mass and all the driver behaviour is encoded in the force $F(t)$. There are many choices you can make for $F(t)$ depending on what you expect drivers to do. We will make the assumption that drivers will tune their acceleration in such a way as to achieve an optimal velocity, which depends on the headway to the car in front.

$$\ddot{x}_n = s\left(v_{\text{goal}}(x_{n+1} - x_n) - v_n\right). \tag{1.5}$$

This is the so-called *optimal velocity model*, and v_{goal} is the *optimal velocity*. The parameter s (in which we have absorbed the inverse car mass M^{-1}) determines how quickly the drivers adjust their speed, it is called the *sensitivity*. The difference $h_n = x_{n+1} - x_n$ is the *headway*.

The driver behavour is completely encoded in the choice of v_{goal}. A much-studied choice is given by

$$v_{\text{goal}}(\Delta x) = v_0\left(\tanh\left[m(\Delta x - b_f)\right] - \tanh\left[m(b_c - b_f)\right]\right). \tag{1.6}$$

This looks more scary than it is (you will make a plot later). Note that v_{goal} becomes zero when $\Delta x = b_c$. The parameter b_c is thus an effective car length; once cars are separated by this distance, they are bumper-to-bumper and they come to a standstill. The v_0 parameter is related to the maximal velocity. Note that here Δx corresponds to the distance between the front bumpers of adjacent cars, not the distance between front and back as it was in question 1.

A simple, stationary solution of (1.5) is given by distributing the cars at equal distance Δx of each other, and giving them a velocity which is equal to $v_{\text{goal}}(\Delta x)$. This is called *free flow*: the acceleration is (and remains) zero. The flow q can now be calculated analytically without doing any simulation, and is given by

$$q = v_{\text{goal}}(L/N)\frac{N}{L} = v_{\text{goal}}(\rho^{-1})\,\rho\,. \tag{1.7}$$

where L is the length of the road and N the total number of cars. Knowing v_{goal} thus immediately gives us the fundamental diagram of q against ρ.

Question 4:

Discuss reasons why it may sometimes make sense to have a traffic model which is fully deterministic (despite the fact that human drivers of course certainly are not).

Question 5:

Discuss the properties which you expect a reasonable v_{goal} to have as a function of the headway. Does v_{goal} as given above in (1.6) prevent cars from crashing into each other? Does it prevent cars from driving backwards?

Copyright © 2015-2017 Kasper Peeters and Bernard Piette; Durham University.

Question 6:

Prove the statements in the paragraph around (1.7), that is, show that the form of x_n discussed there is indeed a solution to (1.5), and show that the flow is as stated in (1.7).

In order to simulate traffic flow which goes beyond the simple free flow, we will resort again to numerical methods. Real world traffic, after all, rarely follows such a highly symmetric pattern. It is highly likely that cars do not start with equal spacing between them, or with velocities which are precisely tuned to the optimal velocity. The question is now what will happen with such random perturbation of the position or velocity. One possibility is that the driver behaviour will correct for these perturbations, such that they get smoothened out and eventually disappear. Another option, however, is that small perturbations progressively get worse, and lead to a different flow pattern.

In the file `traffic_ode_base.py` you will find again a base class `TrafficODEBase` with helper methods. You will create a subclass in which you implement the missing methods.

Coding task 2:

Inspect the class member `__init__` of `TrafficODEBase` to get familiar with the class variables which you will need to use. Notice in particular that `self.x` is an array containing the position of each car. Also inspect the other functions (read at least the docstrings; you do not need to understand how everything works, but you do need to know what the functions do). Now do the following:

- Create a new file `traffic_ode.py` in which you import `numpy` and `traffic_ode_base`. Create a subclass `TrafficODE` which inherits from `TrafficODEBase`.
- Write the `vgoal(self, dx)` method, which computes v_{goal} as in (1.6). Use the `np.tanh` function for the hyperbolic tangent.
- Write the method `plot_free_flow_density(self)`, which uses the method you just wrote, `vgoal(self, dx)`, to make a plot of the analytic result for the flow as derived above in (1.7).

  Estimate the largest car density one can achieve, assuming cars are 5 meters long and use this as the upper bound for the car density on your figure. Evaluate v_{goal} using units expressed in meters and seconds. In your `plt.plot` call, convert the densities and flows so that it plots the flow in cars/minute and the density in cars/kilometer.

  Do *not* add a `plt.show()` call at the end, so that we can overlay this plot with simulated data later.
- Write the `update(self)` method. This is the core of the numerical simulation. It should first compute the headway $x_{n+1} - x_n$ for all cars. It should then use `vgoal(self)` to compute the acceleration a_n given by (1.5), and finally update the velocities and positions according to

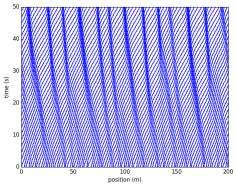
$$\begin{aligned} x_n &\to x_n + v_n \mathrm{d}t + \frac{1}{2} a_n \, \mathrm{d}t^2 \,, \\ v_n &\to v_n + a_n \, \mathrm{d}t \end{aligned} \tag{1.8}$$

  This method should *also* keep track of the number of cars that reach $x_n = L$, and keep a count of them in the `self.detector` variable. Make sure to put the cars 'back at the start' when they reach $x_n = L$!

As in the previous section, you can use the test code below to test your class. It should produce a figure like the one displayed here in the margin. We will return to this figure in one of the questions below. The first argument of the `TrafficODE` initialiser, N, is the total number of cars on the stretch of road. Given a car density (in cars/km), N can be computed using the road length L (be careful about units).

Copyright © 2015-2017 Kasper Peeters and Bernard Piette; Durham University.

```
51 # Test code; only run when this file is not imported as a module.
52
53 if __name__ == "__main__":
54     tr = TrafficODE(100, s=1, m=1, bc=0, bf=2, vmax=1.964, \
55                     road_length=200)
56     tr.setup_cars(random=True)
57     tr.simulate(50, plot=True)
```



In the code above, the parameters have been chosen rather arbitrarily, just to test the code. From now on we will use $b_c = 7\mathrm{m}$, $b_f = 25\mathrm{m}$, $m = 0.12$ and $s = 1.7\mathrm{s}^{-1}$. All the computations are done using meters and seconds, but the figures will display speeds in units that are more familiar (km/h). We will also use realistic values (e.g. a $115\mathrm{km/h} \approx 70\mathrm{mph}$ speed limit as on UK motorways, but feel free to investigate what happens on (some) German Autobahns where there is no speed limit!). The code provided in the next question generates a few useful figures for a few realistic values. Your are encouraged to investigate the problem further for other parameter values.

Question 7:

Run the `traffic_ode_test.py` program. It uses your `TrafficODE` class to simulate various road traffic situations (see the program itself for the values of the parameters used). It produces the following graphs:

a) A graph of v_{goal} as function of the headway. Use this diagram to illustrate your answer to question 5.

b) The results of simulating two traffic flows which differ only in the density of cars. The first one has 60 cars per km, while the second one has 40 cars per km. Plotted is the velocity at a given instant of time, for each car.

   What does the first graph represent, and what can you see on the second one? Does this remind you of situations analysed in the discrete traffic model?

c) A flow-density fundamental diagram for a particular set of parameters which was analysed in [1], for densities of 10 to 80 cars per kilometer. Also plotted is the analytic result for free flow traffic.

   Discuss the graphs of b) in the light of this flow-density graph. Why does the simulation not always agree with the free flow analytic result? Do traffic jams always decrease the flow (with respect to free-flowing traffic) in this model?

Question 8:

It can be proven analytically that the homogeneous free flow is stable when

$$\frac{2v'_{\text{goal}}(h)}{s} < 1\,, \tag{1.9}$$

where h is the headway of the free flow. Given s and v_{goal}, this condition can be used to determine when free flow is stable against perturbations. Find the two values of h for which the left-hand side of the expression above equals 1. Does this agree with your numerical simulation?

4 References

[1] Masako Bando et al., “Phenomenological Study of Dynamical Model of Traffic Flow”, Journal de Physique I (1995) 1389-1399 (available through Google Scholar).

Copyright © 2015-2017 Kasper Peeters and Bernard Piette; Durham University.

5 To submit

You need to submit 3 files,

- one PDF file `essay.pdf` which contains your essay in which you incorporate the answers to all questions,
- one Python file `traffic_discrete.py`,
- one Python file `traffic_ode.py`.

Copyright © 2015-2017 Kasper Peeters and Bernard Piette; Durham University.