Modelling Road Traffic

8.1 Learning outcomes

Mathematics and Physics

- Modelling Discrete systems

Computing and Python

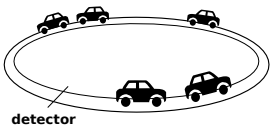
- Iterative methods.

8.2 Introduction

In this project 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

$$q = \frac{\Delta N}{\Delta T} \qquad \text{(fixed position)}\,. \tag{8.1}$$



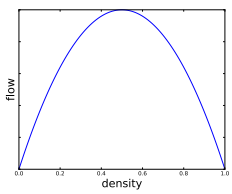
Cars on a road with periodic boundary conditions (circular road) and a detector to measure the flow. Cars drive in one direction only, and cannot overtake.

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

$$\rho = \frac{N}{L} \qquad \text{(fixed time)}\,. \tag{8.2}$$

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 kilometre), 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 8.2, but rather exhibit a sudden change at a particular traffic density.

In the practicals so far, we have mostly asked you to write programs by modifying codes that were given to you. This week, we will ask you to write a program from scratch.

Problem 8.1:

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?[1] 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

$$\ddot{x} = -a_{\text{max}} \,. \tag{8.3}$$

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 $q = v_0/\Delta L$ 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.

Check Point 8.1

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.

8.3 Discrete models

Our first improved model for traffic flow will be a discrete one, in which the road is split up into `road_len` 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.

The first step consists in choosing a car density as a fraction of the maximum density and to distribute cars randomly, with random speed to achieve approximately the desired density (we will provide you with code which does this part of the algorithm).

Once this is set, we proceed by updating the position and the speed of all the cars and we do this in time steps. The speed of the cars are restricted to be integers and correspond to the number of segments each cars move at each step.

Each step is performed 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 a standstill instantaneously. Notice that this rule implies that if 2 cars are in adjacent cells, $h_n = 1$ and hence we set $v_n = 0$ as expected.
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 $x_n \to x_n + v_n$.

It is important that the speed and position of a car at step `k+1` must be based on the position of other cars at step `k`. 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.

Before one starts implementing the model into a program, one must decide how to store the position and the speed of the cars into python objects. There are many way to do this, but we will ask you to use one of the following two:

- Each car is represented by a list of 2 values: `[p,s]`, *i.e.* the position `p` and speed `s` as 2 integers. The cars are then all stored in a list in position order.

  As cars move, when they reach the end of the road, they must reappear at the beginning. This means one has sometimes to move cars from the end of the list to its beginning. The configuration shown on figure 8.1 is then the list

  ```
  [[0,2],[3,1],[4,0],[5,3],[7,1]]  .
  ```

- An alternative is to use an integer array of length `road_len` to describe the road segments. A value of `-1` will correspond to an empty segment and a positive or null value to a car travelling at that speed. An example is drawn below.

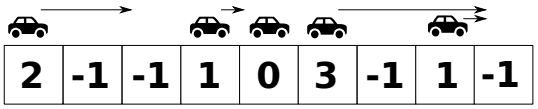


Figure 8.1: A road discretised in segments. The number indicates the velocity. For example, the third car has zero velocity. The fourth car has velocity 3 segments/second which corresponds to 81 km/h. If it does not brake (we will see how we model this shortly) it would crash into the last car. Segments without a car are labelled with $v = -1$ (we do not allow cars to drive backwards, so we can safely use this value).

Regardless of the method you use, after each step, you will append the configuration into a list which will be used at the end of the simulation to generate a space-time graphic representation.

Each representation has its advantages and drawbacks. The array method has the potential to create very fast code using advanced numpy functions, but this is quite tricky. One can solve the problem much more simply by using loops (which we advise you to do here) but when doing so, the program is slower than when using the list method which must also use loops and is quite easy to implement.

Regardless of the method you chose, remember that python does not really make copies of lists and arrays when you write code such as `A=B` and that you should use `A=B.copy()` when you want to make a real copy.

We provide you with 2 programs `tools_array.py` and `tools_list.py`, one for each method, and you should use the one corresponding to the method you have decided to use. They contains 2 functions that you will use:

- `fill_road_randomly(road_len, density, vmax)` : creates a random distribution of car and their speed for a road of length `read_len`, density `density` and maximum speed `vmax`. It returns 2 values: the list or array of cars in the appropriate format and the actual density of cars (which is slightly different from the input value).

- `make_plot(data, density, vmax, road_len, filename='')`: creates a space-time plot of the car positions. The parameter `data` must be a list of all the successive traffic data (the list or array used to represent the car position and speed), while `density` and `vmax` are used for the figure title. When `filename` is not an empty string the figure is also saved in the specified file.

Problem 8.2:

Write a program called `traffic.py` which must import the module `tools_list` or `tools_array` depending on the method you have decided to use.

Write a program called simulate with the following signature:

```
simulate(density, vmax, road_len, p_slowdown, N, Nrelax, filename)  .
```

It must use the `fill_road_randomly(road_len, density, vmax)` function to initialise cars on the road and then iterate the model algorithm. It must first make `Nrelax`

iterations to make the car reach a more or less steady configuration. It must then make `N` iterations saving each configuration in a list.

To avoid using functions with many parameters, we advise you to define a class containing all the relevant parameters and to include in it a function, say `one_step`, that performs a single evolution step. A second class function can then simulates the traffic by calling `one_step` a number of times.

Part of the algorithm requires you to decrease the speed of a car with probability `p_slowdown`. To create a test which is true with probability `p_slowdown` you Will have to import the `random` module and use

```
if random.random() < p_slowdown :    .
```

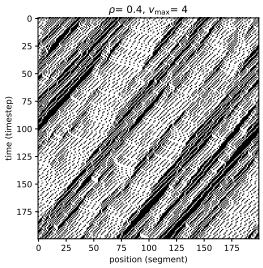
Once the simulation is completed, the program must use the function

```
make_plot(data, density, vmax, road_len, filename)
```

from the imported "`tool`" module to generate a space-time graph of the traffic.

The parameters of the `simulate` function are the following:

- `density` : the car density as a number in the range $(0, 1)$
- `vmax` : the maximum speed of the cars (typically 1,2,3 or 4).
- `road_len` : the length of the road (typically 200)
- `p_slowdown` : the probability to slow down (typically 0.1)
- `N` : the number of iterations.
- `Nrelax` : the number of relaxing iterations.



The program `test_traffic.py` will allow you to test you program. (The last line will generate an error at this stage, just comment it out for now). It should generate a figure of the type shown in the margin.

Programming tips:

- You must implement the 4 steps described on page 2 of these notes as follows: implement step 1 for all the cars, then step 2 for all the cars, then step 3 for all the cars and finally step 4 for all the cars.
- If you use a class, the function `simulate` described on page 3 must be defined outside the class and it must create an instance of the class passing the parameters `density`, `vmax`, `road_len` and `p_slowdown` to the class `__init__` function.
- Depending on the method you decide to use, the road configurations will be arrays or list of lists. The `data` parameter of the `make_plot` function must be a list of these road configurations. Your program must hence generate a list all the road configurations during the simulation. When doing so make sure you make a copy og the array or deep copy of the list, unless your simulator already creates a new list/array for the road configuration at each step.

Check Point 8.2

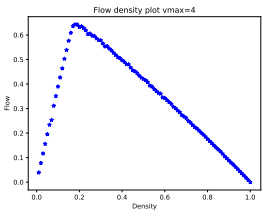
Problem 8.3:

Modify your `traffic.py` program to compute the traffic flow. This is done simply by counting the number of car that move from the end of the road to its beginning (but not during the relaxation). The flow is that number divided by the number of iterations.

Add the function

```
flow_density(dmin, dmax, Nd, vmax, road_len, p_slowdown, filename)
```

to your program. It must scan `Nd` values of the density from `dmin` to `dmax` and perform a simulation on a road of length `road_len` and maximum speed `vmax` and probability to slow down `p_slowdown`. For each simulation, collect the traffic flow value and generate a graph of the traffic flow as a function of the traffic density. If `filename` is not an empty string, the figure must also be saved in the specified file.



Uncomment the last line of `test_traffic.py`. It will allow you to test you program. It should generate the same figure as before as well as a figure of the type shown in the margin.

The program `run_traffic.py` will perform a range of simulations and generate several graphics.

Notice that as the density of traffic increases, the traffic flow increases as well, but once the density reaches a critical value, the flow decreases, something which matches our experience as car passengers or drivers.

Check Point 8.3

8.4 References

[1] Masako Bando *et. al*. “Phenomenological Study of Dynamical Model of Traffic Flow”. In: *Journal de Physique* I (1995), pp. 1389–1399.

