Part1, Particles in a box

1 Particles in a box

To be able to send a satellite out into the vastness of space we will need some sort of fuel for takeoff and also for correctional boosts whilst in space to make sure we reach our destination. In this part we will create a program which calculates the amount of fuel (in kg) we will need for each required boost. But first a small introduction to rocket science and how to applicate it to our project!

In short: This part of the project is very similar to exercises 1A.6 and 1A.7 in lecture notes part 1A. It is recommended to do these two exercises first, then write the slightly more general routine that you need for the project. The goal of this part of the project is simple: you need to write a code which calculates the amount of fuel you need for a given boost Δv. This is exactly what you do in exercise 1A.7, but here you need to make the code valid for any boost Δv, not just to reach escape velocity. During your trip to another planet, your satellite will need to do several correctional maneuevers, and each of these will need fuel. You need the code that you write here later in the project when you have made a planned trajectory with a planned set of boosts so that you can calculate the total amount of fuel you need to bring before launching your rocket.

As you might have read (if not, go back and read it) in **Lecture Notes Part 1A** a rocket engine consists of a fuel tank and an ignition chamber where there are millions and millions of gas particles stacked together. The purpose of the fuel tank is mainly to supply the ignition chamber with fuel (gas particles). Inside the ignition chamber the combustion of the gas creates massive temperatures which in turn creates a massive pressure inside the compartment. The exhaust particles takes part in elastic collisions with the walls of the chamber (remember from FYS-MEK1100, elastic collisions preserve both momentum and mechanical energy). Since the collisions are elastic the particles do not loose energy or momentum inside the compartment. If we now open a hole at the bottom of the chamber (the one facing the ground) particles will leave the compartment through this hole, creating a net momentum loss Δp towards the ground. In order for total momentum to be conserved, the fuel tank (and indeed the rocket/satellite) must gain momentum $-\Delta p$ in the opposite direction. A rocket therefore gets its thrust from the momentum of the exhaust particles leaving the bottom of the rocket. These particles have momentum in the direction pointing towards the ground which gives the rockets momentum up and away from the ground.

This is the general simplified idea of a rocket engine. In this project we will make some shortcuts and assumptions to make sure the project isn't overwhelming but at the same time preserving as much physics as possible.

We will make the following assumptions and short cuts:

1. The density and temperature and thereby the pressure inside the ignition chamber is constant throughout the boost (this means that even though the amount of particles inside the ignition chamber would normally decline towards the end of the boost, we keep it constant).

2. Although the particles escaping would normally be some sort of exhaust particles we will use H_2 gas.

3. The gas does not ignite (we will simply heat up the gas towards the required temperatures, but the particles inside the fuel compartment are the same that leave the nozzle [exit hole]).

4. In order to simulate the rocket engine and calculate the fuel consumption for a given boost, you will, as specified in exercise 1A.7, simulate the boost during a finite time interval (often of a few minutes). However for the actual boost you give the rocket through AST100SolarSystem later on will, for simplicity, be treated as instant. More information about the boost will follow later in the assignment.

5. The particles do not interact/collide with each other inside the compartment. Had they done so this would still not effect the total energy of the tank since these collisions would be elastic.

Part 1.1, simulating particles in a box

To make sure you have a good understanding of how these particles behave, your first assignment will be to create and simulate the particles inside a box. These particles will need velocity distributions as explained in the lecture notes. Make sure that the particles behave as predicted in your simulation before continuing calculating the amount of fuel you need. Below is a short code which you can use to animate the particles in your box in order to check that they behave as expected. To prevent the animation from being too slow, use very few particles (maybe about 100). When you see that the movement of the particles seems correct, you can continue coding, increasing the number of particles to at least 100000 as explained in exercise 1A.6.

```python
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

def plot():
    def update_lines(num, dataLines, lines) :
        for line, data in zip(lines, dataLines) :
            line.set_data(data[0:2, num-1:num])
            line.set_3d_properties(data[2,num-1:num])
        return lines

    # Attach 3D axis to the figure
    fig = plt.figure()
    ax = p3.Axes3D(fig)

    m = 100
    n = ? # number of particles you want to animate
    # the code demands a vector in shape (n, 3 N),
    # where n is number of particles and N are number
    # of iterations.
    # if your arrays are not in this shape, use
    # pos = np.reshape(pos(n, 3, N))
    system = ? # this is the positions of the particles
```

```
    # to be animated. In this code it comes in and array
    # with configuration (n, 3, N) where N is the number
    # of iterations.

    # creates animation data for all your different
    # particles
    data = [i for i in range(n)]
    lines = [i for i in range(n)]
    for i in range(n):
        data[i] = [system[0][i]]
        lines[i] = [ax.plot(data[i][0][0,0:1],
        data[i][0][1,0:1], data[i][0][2,0:1], 'o')[0]]

    # Set the axes properties
    ax.set_xlim3d([0.0, 1.0])
    ax.set_xlabel('X')

    ax.set_ylim3d([0.0, 1.0])
    ax.set_ylabel('Y')

    ax.set_zlim3d([0.0, 1.0])
    ax.set_zlabel('Z')

    ax.set_title('3D Test')

    # Creating the Animation object
    ani = [i for i in range(n)]
    for i in range(n):
        ani[i] = animation.FuncAnimation(fig,
        update_lines, m, fargs=(data[i], lines[i]),
        interval=50, blit=False)
    plt.show()
plot()
```

Part 1.2, calculating the needed fuel amount

We are now ready to calculate how much fuel we will need for a given boost. First of all you will have to create a hole in your box through which the particles can escape and therefore accelerating the rocket/satellite. You will have to count the particles escaping through this hole and their momentum in the direction normal to the surface area of the hole.

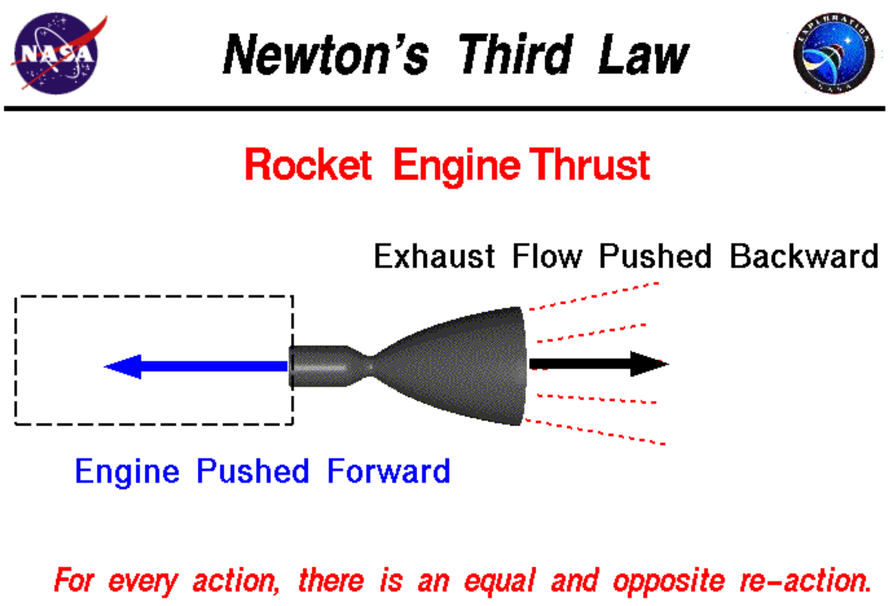
Newton's Third Law
Rocket Engine Thrust
Exhaust Flow Pushed Backward
Engine Pushed Forward
For every action, there is an equal and opposite re-action.

Figure 1: A NASA figure of the fundamentals of rocket engines

Remember one of our assumptions earlier in part1: the density and pressure inside the compartment is constant. The amount of particles that escape the hole as well as their momentum is therefore also constand causing the change in momentum of our rocket engine to be constant. The force in the boost-direction is given by $F = \frac{dp}{dt}$. A crucial point here is that the particles behave the same way inside a small compartment as they do inside a big compartment. This means you can (to make the simulations faster) create a small fuel compartment and find out how much acceleration it gives you in the boost direction. You can then just add enough equal compartments (the equivalent of having 1 big one) until you get the required acceleration.

You have now (hopefully) grasped how to accelerate your rocket. What remains now is how much fuel you will need.

The goal of this part is to write a code to calclate the amount of fuel needed to give the rocket a given boost Δv. To do this you will have to split up your boost into small time intervals. In exercise 1A.7 you calculate the amount of fuel that you need in order to reach escape velocity in 20 minutes. Here you need to experiment to find a way to calculate the time and the total fuel needed for a general velocity change Δv. You may adjust gas density, temperature, total boost time and the size of the hole in order to obtain your goal. The less fuel and the less acceleration time you manage to get, the better. But it is difficult to get both these measures low at the same time. You will need to find a compromise.

In the end you should be able to calculate how many kilograms of fuel you need to boost a rocket with mass 1100 kg. To make sure your calculations are correct there's a module inside the
AST1100SolarSystem class which checks this for you (you will be given more information on how to use this class in the next part). All you have to do is to enter the following lines at the end of your code and send in the required variables:

```
from AST1100SolarSystem import AST1100SolarSystem
system = AST1100SolarSystem(**seed**)
system.massNeededCheck(num_boxes, boost [AU/yr],
dp/dt [kg*m/s^2] (per box),
num_particles_pr_sec_from_one_box, your_answer [kg])
```

this method tells you if your calculations are correct (within some tolerances) or not, it does not give you the right answer. For the proper launch to reach escape velocity, you will need to include the mass of all your fuel in addition to the mass of the rocket. Remember to reduce the total mass of the rocket for each small boost in time step `dt`.

For the first launch from your home planet, you can use

```
escapeVelMovie(self, deltaV, dt)
```

to see a video of you launch. This method takes the arguments `deltaV` (the list of velocities for each time step `dt` in your calculation). It returns an xml file to be loaded in MCAst showing you the trajectory of your satellite as you send it out into space. This function is made for exercise 1A.7 where the task is to reach escape velocity. As explained in that exercise, the output of the video depends on whether your velocity is smaller than, equal to, or larger than the escape velocity. In this project, you may choose to start with a different initial boost. It is therefore not important for you to check whether you reach escape velocity or not.