Satellite project, AST 1100

Interlude: The Habitable zone

We will start with some considerations to help you decide which planet to land on and what you need to bring of solar panels to have electricity for your lander when you arrive at your destination.

Problems:

Exercise I.1: Find a formula which gives you the total area of the solar panels that your lander unit needs during day time on your destination planet. The formula should depend on properties of your star as well as some properties of the planet and the distance to the planet. Make the formula general so you can use it for the different planets in your solar system. Assume that the lander needs 40W during daytime to run its instruments and that the solar planels have an efficiency of 12%. Use exercise 1D.3 in part 1D as an assistance to arrive at the answer.

Exercise I.2: Find a general expression for the surface temperature of a planet in your solar system, as a function only of the surface temperature T_* of your star, the radius R_* of your star as well as the distance r_p between the star and the planet. Use exercise 1D.4 in part 1D as a help.

Exercise I.3: If we assume that the surface temperature of a planet has to be within $260 - 390\,\mathrm{K}$ in order to sustain life[1]. Find an expression for the minimum and maximum distance from the star where a planet could sustain life, using the (very simplistic) assumptions used in the last problem. If we define this distance range to be the *Habitable zone*, what is the habitable zone of our real solar system? Your star system? Which of your planets fall within the habitable zone?

Part 3: Simulating the satellite trajectory

In this part you are preparing ourselves for a satellite trip to another planet. Remember we are still in the planning phase of the mission and are only planning for what we THINK will happen. Later on, in part 5, you will be tasked with launching the actual satellite and adjusting the plan according to any unforeseen events which occur. Even later, in part 6, you will then be asked to attempt to land the satellite, but for now, let's see what happens.

[1] This is the temperature range of liquid water ($\pm \sim 15K$).

The ultimate objective is to get your simulated satellite sufficiently close to the target planet to be able to perform an orbital injection maneuver, placing the satellite in a stable orbit.

The first task is to choose the target planet. In part 2 you already had a look at your different planets using SSView. Go back and look at your solar system again: try to identify your planets using the information you have about the distance from the star combined with the actual distance you see in SSView. Then use the solutions of the above problem to find which of your planets are in the habitable zone (in case you want to try to visit a planet with life, but an ice planet or something else could of course be equally exciting).

When you have chosen your planet, use the formulas you deduced above to find (1) the size of the solar panels you need to bring and (2) the average surface temperature of your target planet. Note that the size of the solar panels may become unrealistically big for some of the outer planets, but never mind. Now we will start finding how to launch and then further boost the satellite in order to arrive safely.

After choosing a planet, send an e-mail to the group teacher with your seed and your choice of destination planet.

3.1 Simulating the satellite trajectory

First of all we will simulate the satellite trajectory. Your task is to find a set of instructions to give the satellite which results in it hitting (or getting very close to) another planet.

In the next parts of the project, you will start communicating with your satellite. There are two different types of instructions you can give the satellite:

(i) Launch: Use large booster rockets to give the satellite an initial acceleration needed to lift it from your home planet and achieve escape velocity. This command is only given once, at the start of the mission (does not need to be at $t = 0$).

(ii) Boost: Use the satellite's rocket engines to change the velocity of the satellite with a vector $\Delta\vec{v}$. This command is given mid-flight, and can be issued multiple times during the mission.

You will need to use these two instructions in later parts, in this part your task is to calculate how many boosts you will need, when they should be executed and the acceleration vectors you need for the launch and for each boost. Then in later parts you will need to insert these numbers in the instructions to the satellite. In later parts you will also need to go back to this part to make corrections to your orbit: When your satellite are in space you will measure correct positions and velocities which in general will deviate slightly from the ones which you calculate in this part. For this reason you will later need to reuse the code you develope here using slightly different input positions and velocities.

The actual calculations you will do are very similar to those you did in part 2, but now you only need to calculate the trajectory of the satellite, and not the planets. We make the following set of simplifying assumptions:

- We include the gravitational force on the satellite from all the planets and the star, but we neglect the gravitational force on the planets and star from the satellite.

- We neglect the atmosphere of the home-planet when launching the satellite.
- Moons are completely neglected (they are just decorations).
- The launch until escape velocity (which you already simulated in part 1) happens over several minutes and propels your spaceship in a straight line up *above a fixed point on your home planet surface. The point of launch in time and space is yours to decide. After the rocket has reached escape velocity, the satellite is considered "in space".*
- *All effects of using the engines on the satellite* in space are instantaneous. That is, using the engine results in an instantaneous change in velocity. This in not too unreasonable since a rocket engine can work in seconds or minutes, while the whole satellite trip takes months or years. Although we treat the boosts as instantaneous, you will need to count with a finite boost time when calculating the fuel as explained in part 1.
- The satellite moves in the same plane as all the planets (the x-y-plane).
- Remember to calculate the amount of fuel you need for each boost (including the launch). You will need this total amount of fuel later on in part 5. It's a good idea to start with too much fuel, then (optionally) adjust down the amount after reaching the destination, rather than starting with too little and having to readjust the launch several times during your simulations.

You are solving the following equations:

$$\frac{d^2\mathbf{r}_S}{dt^2} = -\sum_i \frac{Gm_i}{r_{iS}^3}\mathbf{r}_{iS}, \tag{1}$$

where i runs over all the planets and the star, $\mathbf{r}_S$ is the position of the satellite, m_i is the mass of planet (or star) i and $\mathbf{r}_{iS} = \mathbf{r}_S - \mathbf{r}_i$ is the relative position vector between planet i and the satellite. Solve these equations using your favourite integration method, like you did for the planets in part 2. Use the positions of the planets that you found in part 2 (we recommend you use the `scipy.interpolate` library to interpolate planet positions because you might need a lot more time steps).

You need your satellite to get close enough to the target planet for a stable orbit to be possible. The minimum distance needed to achieve this varies depending on the characteristics of the planet, but the minimum distance will typically be of the order of magnitude of $1.5 \cdot 10^6$ km or 0.01 AU. The following exercise will help you calculate the exact distance.

Exercise 3.1.1 : **(optional for those working alone!)** Show that the force on the satellite from the target planet is k times as large as the force on the satellite from the star if the distance from the planet is

$$r = R\sqrt{\frac{m}{M}\frac{1}{k}}, \tag{2}$$

where R is the satellite–star distance, m is the mass of the planet, while M is the mass of the star. If we set $k \gtrsim 10$, this will yield a distance

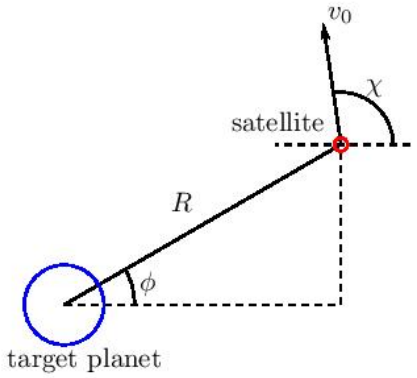

Figure 1: The geometry of the planet–satellite system at a time when the distance between the two is R, with R hopefully small enough to achieve a stable orbit after an orbital injection maneuver is performed. The angle ϕ denotes the angle between the x-axis and the planet-satellite vector. The angle χ denotes the angle between the velocity of the satellite relative to the target planet at this point, v_0, and the x-axis. Note that if we regard the closest approach between the satellite and the planet, the relative velocity will be perpendicular to the planet–satellite vector, i.e. $\chi = 90° + \phi$. However, since we are using finitely many time steps in our simulation, this will in general not be true of *our* "closest" approach.

at which the forces on the satellite are dominated by the force from the planet, and a stable orbit is possible. A larger k will mean a more stable orbit, but $k = 10$ is the threshold.

Exercise 3.1.2 : **(optional for those working alone!)** Show that the velocity you need in order to get into a stable circular orbit around the target planet is

$$v_{\text{stable orbit}} = \sqrt{\frac{M_{\text{planet}} G}{R}},$$

where $v_{\text{stable orbit}}$ is the velocity *relative to that planet*, and we neglect the star and the other planets. Denoting ϕ and χ as shown in Fig. 1, show that in order to achieve a stable circular orbit clockwise around the planet (flip both signs for counter-clockwise orbit), the velocity needs to be

$$\vec{v}_{\text{stable orbit}} = \vec{v}_{\text{so}} = \big(-v_{\text{so}} \sin\phi,\ v_{\text{so}} \cos\phi \big).$$

Show also that the change in velocity needed to achieve this is given as

$$\Delta\vec{v} = \big(-v_{\text{so}} \sin\phi - v_0 \cos\chi,\ v_{\text{so}} \cos\phi - v_0 \sin\chi \big).$$

(Hint: Centripetal acceleration.)

NB! The velocity v_0 and angle χ are as seen from the reference system of the planet. What are the expressions for these, given the satellite initial velocity $\vec{v_{\text{sat}}} = (v_{\text{sat}x}, v_{\text{sat}y})$, and planet velocity $\vec{v_{\text{pl}}} = (v_{\text{pl}x}, v_{\text{pl}y})$, both

relative to the star. To further complicate the process, recall that you only have a list of positions for the planets at different times. How would you proceed to numerically find $\vec{v_{\text{pl}}}$ from the list of positions $\vec{r_{\text{pl}}}$?

Hints :

- Make your code general, with the option to start your satellite from any point in time and space. This is a good idea because in part 5, you will be able to get the position and velocity of the satellite. If at some point they do not match, it will be necessary to start the simulation with these parameters and find a new set of boosts to correct the stray course.
- Use smaller time steps for the simulation of the journey than you did for the simulation of the planets. Your spacecraft is supposed to move and accelerate faster than the planets do. A good idea is to do convergence testing; make a rough satellite trajectory passing close to the target planet, then reduce the time steps and rerun the simulation. Repeat until the trajectory does not change due to numerical time resolution.
- **If you work alone** You do not have to actually simulate the last boost (orbital injection maneuver), you only have to compute how large it has to be and in what direction in order to achieve a stable orbit. The satellite orbiting the target planet and the landing is handled separately, in parts 6 and 7.
- **If you work in a group** you need to simulate the last boost and the subsequent orbit of the satellite around the target planet, you will have to lower the time step drastically once you get close (adaptive time steps).
- In order to get the positions of the planets at an arbitrary time using the list of positions and times from part 2 you can use the interpolation function in python:

```
1  import numpy              as np
2  import scipy.interpolate as inter
3
4  # Import positions and times calculated in part 1.
5  inFile           = open('planet_positions.npy','rb')
6  # if you saved both positions and times, use this:
7  planetPos, times = np.load(inFile)
8
9  #planetPos has shape (2, N_planets, time_steps)
10
11 # The following call makes the interpolation function.
12 # The last index of planetPos is interpolated (time).
13 positionsFunction = inter.interp1d(times, planetPos)
```

Now you have a function that interpolates based on your original list and returns positions for any time. If you want the positions of the planets at any time t_x you just call the function:

```
1      positions_at_t_x = positionsFunction(t_x)
2      #positions_at_t_x has shape (2, N_planets)
```

- It is not entirely trivial to get the satellite close to another planet, but with a bit of playing around, using different combinations of boosts and some trial and error, you should be able to get fairly close. To do this playing around you can for example continuously print the distance between the satellite and the target star for every 1000th time step.
- Because this is just your own simulation of the trajectory, there is no "test" to prove you passed this part. You yourself decide how precise you want to be before moving on to the next parts. Please remember that the real mission might have some differences, such that time spent perfectionizing the last percent precision will effectively be wasted.