Data Analytics Assignment-1: Mars Orbit

G. Aravind

August 26, 2021

1 Formulation of the problem

1.1 Model assumptions

- The Sun is the centre of our coordinate system (Sun is at the location (0, 0) in the ecliptic plane).
- Mars follows a circular orbit, with centre angle c and distance 1 from the Sun. That is, the centre of the orbit is at $(\cos c, \sin c)$ in cartesian coordinates.
- The radius of the Mars orbit is r.
- The equant is located at (e_1, e_2) in polar coordinates, or $(e_1 \cos e_2, e_1 \sin e_2)$ in cartesian system.
- The reference longitude is given by angle z with respect to the Aries.
- The angular velocity of Mars around the equant is s degrees per day.

1.2 Derivation of the angular error

The model of Mars orbit can be visualised as in Figure 1 (only the first opposition is drawn for simplicity). Here, P_1 refers to the position of Mars with respect to the equant, and P_2 is the opposition observation. The line from P_2 to Sun (Line 2) makes an angle l with Aries, where l is the heliocentric longitude of the first opposition. And, P_1 makes an angle $l + \delta$ with Aries (Line 1), so basically we want to find this angular error δ.

Since the centre of the Mars orbit is $(\cos c, \sin c)$, the equation of the circular orbit can be written as follows:

$$
\begin{aligned}
&(x - \cos c)^2 + (y - \sin c)^2 = r^2 \\
\Rightarrow &x^2 + y^2 - 2x \cos c - 2y \sin c + 1 - r^2 = 0
\end{aligned}
$$

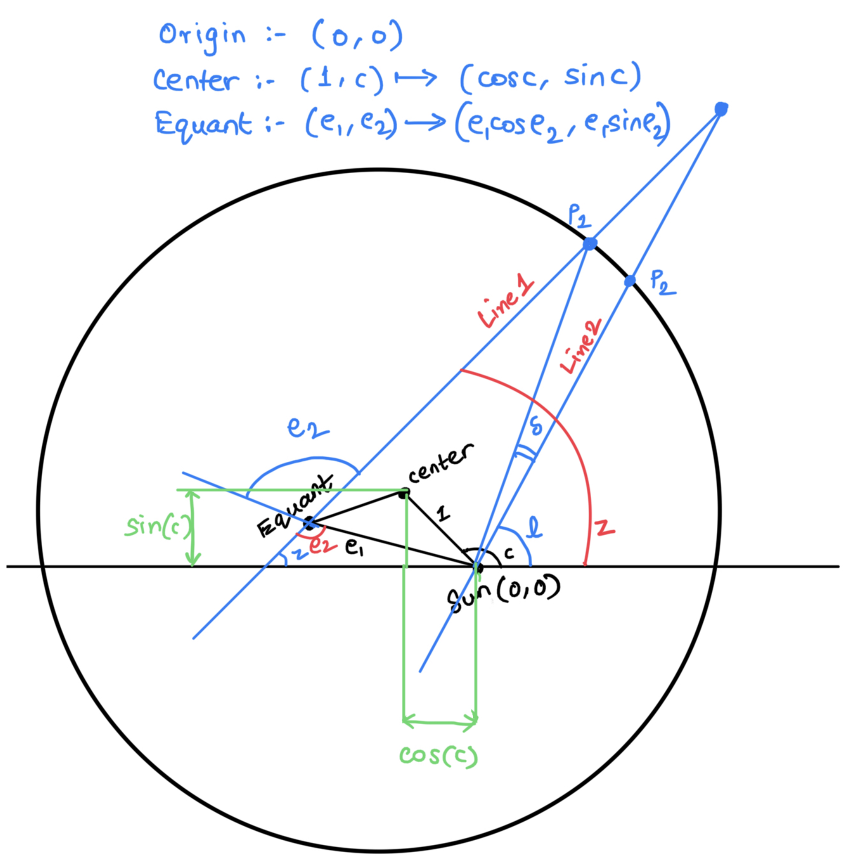

Figure 1: Example of a parametric plot $(\sin(x), \cos(x), x)$

The equation of Line 1 in Figure 1 can be written as follows:

$$
\begin{aligned}
&y = m_1 x + c_1 \\
\Rightarrow &y = x\tan z + c_1 \text{ (Since the line makes angle } z \text{ with Aries)} \\
\Rightarrow &e_1 \sin e_2 = e_1 \cos e_2 \tan z + c_1 \text{ (Since equant also lies on the same line)} \\
\Rightarrow &c_1 = e_1 \sin e_2 - e_1 \cos e_2 \tan z
\end{aligned}
$$

Now we want to find the point of intersection of Line 1 with the orbit (P_1) in order to compute δ. Substituting $y = x\tan z + c_1$ in the equation of the circular orbit, we get:

$$
\begin{aligned}
&x^2 + (x\tan z + c_1)^2 - 2x\cos c - 2(x\tan z + c_1)\sin c + 1 - r^2 = 0 \\
\Rightarrow &x^2(1 + \tan^2 z) + x(2c_1\tan z - 2\cos c - 2\tan z \sin c) + (c_1^2 - 2c_1 \sin c + 1 - r^2) = 0
\end{aligned}
$$

This is a quadratic equation of the form $ax^2 + bx + c = 0$, where

$$
\begin{aligned}
a &= 1 + \tan^2 z \\
b &= 2c_1 \tan z - 2\cos c - 2\tan z \sin c \\
c &= c_1^2 - 2c_1 \sin c + 1 - r^2
\end{aligned}
$$

We solve this quadratic equation in order to get the x and y coordinates of the point P_1. Now, since the Sun is at the origin $(0, 0)$, the angle $(l + \delta)$ is nothing but $\tan^{-1} \frac{y}{x}$. Hence we can get δ by subtracting l from this angle.

$$\delta = \tan^{-1} \frac{y}{x} - l$$

Apart from the formulation described here, we also have to handle some additional corner cases while writing code. For e.g., the above quadratic equation can have two roots (as there can be two points of intersection between a line and a circle), which we denote by (x_1, y_1) and (x_2, y_2) in the code. So in the code, we compute two angular errors δ_1 and δ_2, and we choose the minimum of these two as our final δ. More details on the code structure is explained in the next section.

2 Structure of the code

After reading the data from CSV file using pandas, the code has been divided into functions for simplicity, each of which are briefly described below.

- **get_time_diff:** This function computes the time differences between consecutive oppositions from raw data. This is needed, because time difference between oppositions multiplied by angular velocity would help to compute the equant angles for Mars at all oppositions.

- **get_longitudes:** This function computes the heliocentric longitudes of Mars at oppositions, using the various columns given in the csv file. These longitudes will be used to compute δ later.

- **get_data:** This function takes the file path as string input, and internally uses the *get_time_diff* and *get_longitudes* functions, and vertically stacks together the time-differences and longitudes in a single variable called *data*, which is returned. This *data* can now be passed on to other functions as parameters.

- **find_delta:** Given the coordinates of point-of-intersection P_1 as (x, y) and corresponding heliocentric longitude, this function computes the angular error δ (difference between $\tan^{-1}(\frac{y}{x})$ and l). While computing $\tan^{-1}(\frac{y}{x})$, I have also taken care of the cases of angles lying in each of the four quadrants.

- **MarsEquantModel:** This function takes the model orbit parameters c, r, e_1, e_2, z and s, and computes angular error δ for each opposition, using the derivation which is explained above. The function returns the errors of each opposition, as well as the max error of all these errors.

- **bestOrbitInnerParams:** This function will take the values of r and s as inputs, and searches for the best other parameters for these fixed values of r and s over a feasible search space, using the previous function *MarsEquantModel*.

- **bestS:** This function takes a fixed value of r as input and searches for the optimal value of s using gridsearch, and inside the code, it uses our previously defined function *bestOrbitInnerParams*. It returns the optimal s and corresponding errors.

- **bestR:** This function takes a fixed value of s as input and searches for the optimal value of r using gridsearch, and inside the code, it uses our previously defined function *bestOrbitInner-Params*. It returns the optimal r and corresponding errors.

- **bestMarsOrbitParams:** This function takes only the oppositions data as input, and searches for suitable values of both r and s using gridsearch, and returns the approximate optimal parameters which result in lowest angular error δ.

3 Observations

- Based on Moodle discussions and my own understanding of the problem, I initially estimated the model parameters as: c = 150°, r = 9.0, e_1 = 1.4, e_2 = 150°, z = 54°and s = 360/687°per day. For these parameters, the maximum angular error δ I got using *MarsEquantModel* function was 3.98°.

- By using the functions for next 4 subproblems, I estimated the optimal parameters for my Mars orbit model. My final estimated optimal parameters are: c **= 139.0,** r **= 8.732,** e_1 **= 1.60,** e_2 **= 148.70,** z **= 56.47,** s **= 360 / 687**. And the corresponding maximum error is **0.40878 degrees (24.527 minutes)**. The corresponding errors for each opposition from 1st to 12th opposition are: [0.244, 0.308, 0.408, 0.394, 0.380, 0.408, 0.408, 0.003, 0.062, 0.048, 0.063, 0.036] (in degrees).

- I also tried doing more exhaustive grid search with more precision in parameters, but the estimated execution time for such large search-spaces was more than 24 hours, and it was difficult to run the code for such long durations on my personal laptop (sometimes there would be interruptions in the code in the middle of execution), hence I narrowed down my search and slightly reduced the precision in order to be able to practically run the code. Using these settings, the least angular error that I have managed to get is 24.5 minutes.

4 Instructions to run the code

- First run the *get_data* function with the name and path of the csv file as input, and store it in a variable *data*.

- Pass on this *data* as parameter along with other numerical parameters in order to test the subsequent Mars orbit functions.