Assignment 2: Mars Orbit Model

Subhojyoti Khastagir
SR. no.: 04-04-00-10-51-21-1-19446
M. Tech. - CSE
Department of Computer Science and Automation
subhojyotik@iisc.ac.in

1 Introduction

The objective is to make an estimate of the orbit of Mars around the sun using the equant model. Although not a very accurate model, but still a significant step towards finding out the more appropriate ellipse model. In this model, we assume that the sun is off center from the center of the circular orbit. This off-center still does not explain the varying angular speed, so we assume yet another point called the equant about which the angular speed of Mars is assumed to be constant.

2 Data

We are provided with the longitudes and times of twelve instances of oppositions between the Sun, Earth and Mars. Since during an opposition, the geocentric and heliocentric longitudes are same, we can trace the heliocentric location of Mars using the longitude data. With the time data and the assumption that the angular speed of Mars is constant about the assumed equant point, we can similarly trace the location of Mars about the equant. With these two sets of locations, we can then pinpoint the twelve locations of Mars corresponding to the twelve oppositions

3 Calculation

After obtaining the twelve locations of Mars, we calculate the error in observation by finding out the angle between the equant point and the given longitude for each of the observation. The maximum absolute error is the objective which has to be minimized.

To calculate the equant point, we have to find out the point of intersection between the assumed orbit and the line through the equant. Let (c_x, c_y) be the point representing the center of the orbit of Mars, and (e_x, e_y) denote the equant point. The radius of the orbit is given to as r units.

$$c_x = 1 \times \cos c, \quad c_y = 1 \times \sin c$$
$$e_x = e_1 \times \cos e_2, \quad e_y = e_1 \times \sin e_2$$

Let us consider an arbitrary ith opposition data point. Let the angle of the equant line from the Sun-Aries reference line be θ. This value can be obtained from the opposition time data and z which is the equant-0 line. Also, let us denote $\tan\theta$ by λ.

The equation of this line will be

$$y - e_y = \lambda(x - e_x)$$

We also know the equation of the orbit of Mars from the given center point and radius

$$(y - c_y)^2 + (x - c_x)^2 = r^2$$

Solving these two equations simultaneously, we get

$$x^2(1+\lambda^2) + 2x(\lambda e_y - \lambda^2 e_x - \lambda c_y - c_x) + (e_x^2\lambda^2 - 2e_x\lambda e_y + e_y^2 - 2e_y c_y + 2e_x\lambda c_y + c_y^2 + c_x^2) = r^2$$

Now, we apply Shridhara Acharyya's formula to get the two values of x, lets say x_1 and x_2. Then using those two values and putting them in the equation of the line, we get two corresponding values of y, lets say y_1 and y_2. Hence we have obtained two points of intersections (x_1, y_1) and (x_2, y_2).

Out of these two points, we have to consider only one, which can be decided by checking the magnitude of angle θ. It θ lies in the first or fourth quadrant, we will consider the point which lies on the right side (has a greater x value), otherwise we consider the point on the left side (has a smaller x value).

Using this point of intersection and the longitude line, we calculate the difference which is the error. We get twelve such errors out of which we aim to minimize the maximum error.

4 Solving the Mars model

4.1 Brute force

The naive approach in solving the problem is to perform an exhaustive search over all possible combinations of parameters to get the best combination. This approach takes very long time (more than 30 mins) so this approach is not feasible.

4.2 Using scipy optimize function

The scipy library in python provides functionality to perform optimization of scalar values using the minimize function. The only catch is that it requires an initial guess from which it optimizes further using an objective function.

It takes a fifth of a second to run, and after lots of trial and error with the initial guesses of the parameters, gives a max error of 2.38 degrees. The optimal values of parameters are given below

	c	r	e1	e2	z	s
Initial guess	45	7	2	145	60	0.524
Final values	45.00611795	7.63076426	1.35503642	144.99245996	59.67892577	0.52351875

4.3 Using a combination of both approaches

The brute force solution searches the whole space of input parameters but takes a lot of time. On the other hand, the scipy optimize functionality runs lot faster but requires an initial guess. Using the two approaches together can help in overcoming the huge time requirement as well as the initial guess requirement.

The idea here is to use minimize function for optimizing the parameters but instead of providing an initial guess, we can perform a sparse search over the parameter space to come up with many different initial guesses and take the best among them. Using sparse search reduces the overall time requirement, and by using minimize function, we make sure to reach an optimal value.

Although it takes less time than a brute force exhaustive search, still the time taken is significantly high.

4.4 Sequential optimization instead of nested search

This approach arises from the fact that varying only one parameter keeping the others fixed will be faster than varying all parameters in a nested manner. Linearly searching over the s and r parameters one after the other using the $bestS()$ and $bestR()$ functions gives a good enough estimate for the two parameters. The other parameters are guesses in the manner described above.

Using a random initial guess of $c = 10, e_1 = 1, e_2 = 10$, we first run an exhaustive search over z using a step size of 0.5 degrees. Then using this optimal value of z, we search over e_1 in the range of 0 to 3 with a step size of 0.02. Then using the optimal value of z and e_1, we obtain an estimate of e_2. Finally we obtain an estimate of c using the optimal guesses of the previous three parameters. Now we have an optimal guess of the four parameters c, e_1, e_2, z. The whole procedure can be repeated few times to obtain an even better guess.

Using the above obtained optimal guesses, it can now be used to get the final optimal values using the scipy minimize function. This method takes a feasible time of 7.4 mins (Ubuntu 20.04.2 64-bit running on a 12-core Intel Core i5 11th gen with 16GB RAM) and produces the following optimal values for the parameters with a max error of 0.1551 degrees

c	r	e1	e2	z	s
145.2571	7.2437	1.3443	148.8935	55.8248	0.5241

5 Conclusion

Even though the data has only twelve data points, the parameter search space is so large that it is not feasible to perform an exhaustive search, which would have been the ideal method for finding a global minima of the error, hence different methods and their combinations had to be used to speed up the process and find the optimal value without exhaustively searching the parameter space. The final method used was to find the optimal guesses for r and s first and then using the optimal guesses to obtain the other parameters by sequential optimization keeping other parameters fixed, and finally using the optimal initial guesses to come to a final optimal value using the library function.

The diagram of the optimal orbit is given below

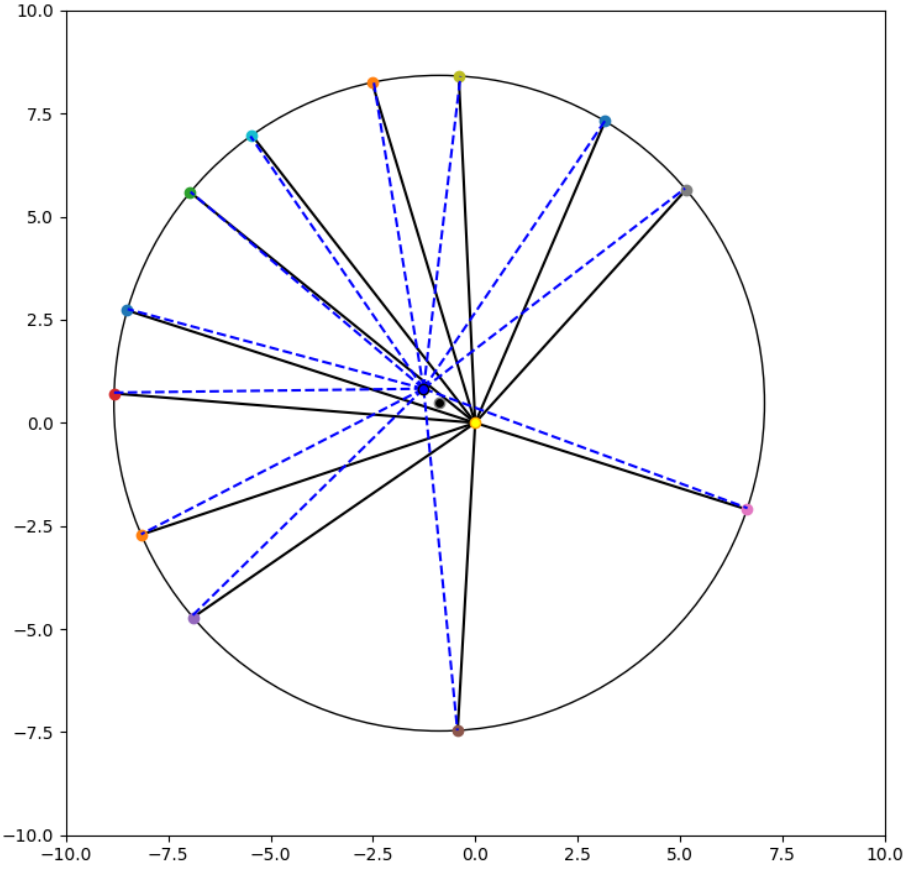

Figure 1: Optimal orbit