Abhijeet Jawlekar | SR No.: 06-18-01-10-51-21-1-20128

Report

Assignment-2: Mars Orbit

Implementation summary:

Data preprocessing:

To convert given date (year, month, day, hour, minute) to timestamp I used datetime module, but date time is defined for year more than 1900 but our data starts from 1580 so I added 400 to its year column as 400 is divisible by 4 so it wont affect in number of time between readings could be arise due to leap year issue.

Then relative time gap between the readings is calculated using timestamp method of datetime and this time gap converted into days.

Angle of each opposition is calculated using angle=ZodiacIndex*30 + Degree + Minute/60 + Second/3600

MarsEquantModel (variables,r,s,times,oppositions):

A function named **MarsEquantModel(variables,r,s,times,oppositions)** is defined which will take all parameters and will return error and max(error) that is discrepancy where variables=[c,e1,e2,z]

Logic: by knowing all parameters I can find the point of intersection of circle of radius r and a line at angle zi from equant now solving simple trigonometric 2nd degree equation one can estimate position of mars (x, y) co-ordinate where sun is at origin now knowing the estimated position and actual latitude of opposition, I can find difference between then and it would be my discrepancy.

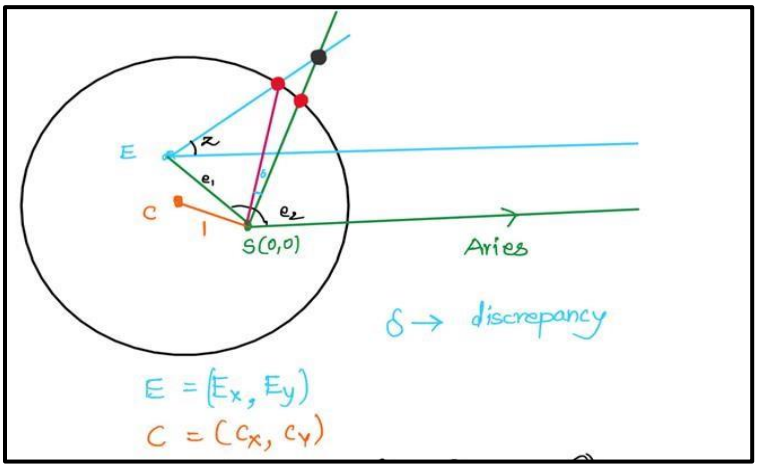

Fig: figure showing discrepancy or error

bestOrbitInnerParams(r,s,times,oppositions):

A function is defined which will take the initial guess of variables and further fine tune it by optimize.minimize function using the 'Nelder-Mead' method.

bestS(variables,r,times,oppositions):

This function will search for best s minimizing error given by MarsEquantModel by discretely searching for s in range (360/688, 360/686) as we know approximate revolution time of mars around sun is 687 days.

Now this discrete search uses technique like bisection with generating 100 linearly spaced S values and get one with minimum again discretize it around that minimum s value again fine tune it. This done repetitively for 3 times.

bestR(variables, s,times,oppositions):

Given all other parameters this function finds the intersection of dotted lines that is estimated line of opposition from equant and solid line that is line of opposition from sun (from given longitudes) this is done by solving two equations one for x co-ordinate of intersection and other for y co-ordinate of intersection of two vectors having length say j and k. now after getting intersection point its distance from center is calculated. Doing this for all oppositions minimum and maximum radius is found and did the discrete search within this range.

Now this discrete search uses technique like bisection with generating 100 linearly spaced r values and get one with minimum again discretize it around that minimum r value again fine tune it. This done repetitively for 3 times

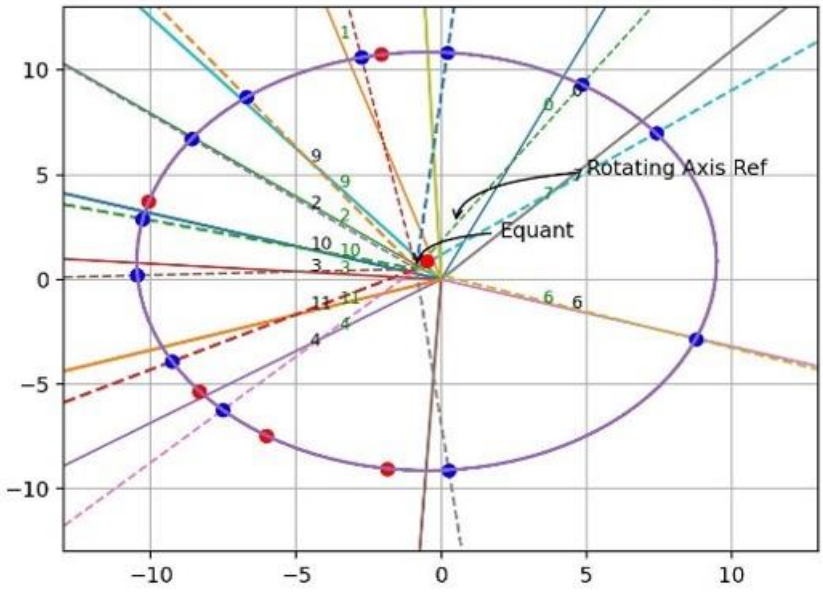

Fig: representative figure showing intersection of solid and dotted lines.

Variables,r,s,errors,maxError = bestMarsOrbitParams (times,oppositions)

Initialized the value of R=10 and S=(360/687)

the wrapper function is created which discretely search for variables that is [c,e1,e2,z] (keeping r and s constant) by scipy.optimize.brute initially I searched for very high ranges and providing quite high step slice I used *(ranges = (slice(0,180,5), slice(1,2,0.5), slice(0,180,10), slice(50,70,5))* to get in initial estimates of parameters then I got variables=[155,1.5,150,55] and giving max error around 3.5 degrees.

It observed that final error value is sensitively dependent on initial guess so rather than going with initial guess values got from brute search strictly I generate other random initial values of parameters around the values I got from brute search

generated parameters randomly using numpy.random.uniform to get values of parameters to start with. For example, random value of s generated around (360/687), random value of r around 10 and similarly random value for c around 155, e1 around 1.5, e2 around 150, z around 55. uses functions bestOrbitInnerParams, bestS, bestR iteratively two times and save the error and parameters value in a list. Did this till error falls with in 4 min that is 0.07 degrees approximately.

Results:

Finally running bestMarsOrbitParams (times,oppositions) one can find best parameters for which discrepancy error falls within 4 min that is 0.07 degrees for following parameters:

```
Fit parameters: r = 8.4784, s = 0.5241, c = 149.6408, e1 = 1.5754, e2 = 14
9.0660, z = 55.8946
The maximum angular error = 0.0641 =3.846 min
```

```
Errors=[0.032711046031991486,0.05564074042780476,0.06412051644599615,0.047
67315961657914,0.06412073035923527,0.050889923393642,0.035762165160804216
,0.0020049685092331515,0.02932214556597046,0.0370940883780122,0.0641206531
3949711,0.040208999231452935]
```

Remark: As algorithm uses brute search and randomized technique it takes quite large time to reach for best parameters to error fall within 4 minutes.