E0 259 - Assignment 2

Shankaradithyaa Venkateswaran
Sr no: 22190

September 13, 2024

Disclaimer

I have not followed the template provided in the MarsAssignment.docx and I have used Github Copilot to help in writing my code. As I have reported maximum error along with the sign, I have edited the Assignment2.py template line 39 to check the equality with abs(maxError) instead of maxError as I have included sign with maxError.

Output

After running the code, the output is as follows:

$$
\begin{aligned}
r &= 8.675199386469073 \\
s &= 0.5240810650333936 \\
c &= 148.92895420324808 \\
e1 &= 1.6120197451470566 \\
e2 &= 148.9318624984065 \\
z &= 55.8389212681124 \\
errors &= [-0.035706662541514333, 0.008367130610537288, \\
&\quad 0.019686529582031653, -0.001140502720488712, \\
&\quad 0.03149420503592637, -0.024783850906203497, \\
&\quad 0.009077761891205682, 0.023529392025054108, \\
&\quad 0.0029609002745019097, -0.0011668646487805745, \\
&\quad -0.0330643882880679, 0.0007198958661263077] \\
max\_error &= -0.03570666254151433
\end{aligned}
$$

I have reported the maximum error along with its sign in the output.
It took a total of 181 minutes and 10.7 seconds to run the entire exhuastive search. Actually the function had converged a while back, within 20 minutes due to the efficacy of the minimize function, hence, the rest of the running time is unecessary. Following this result I made an optimization that if a certain number of iterations passed without any improvement in the maximum error I would end the loop.
After the optimization the running time is 23 minutes and 38.9 seconds.

Implementation Summary

I have used the template provided in the assignment as a base. In it, I have called two files with helper function with the same names as the functions mentioned in the template.

Extracting Data
The *get_data.py* file contains two functions which take in the data in the form of a 2d numpy array and use the *get_times* function to output the times of the twelve oppositions in days; and the *get_oppositions* function to output the latitudes and longitudes of the twelve oppositions.
It extracts the data given that the columns are in the order of Year, Month, Day, Hour, Minute, ZodiacIndex, Degree, Minute, Second, LatDegree, Lat-Minute.

Now this data is stored in times and oppositions repectively, and then passed through the $bestMarsOrbitParams$ function. I have written this function along with several helper function in the *marsorbit.py* file which is also imported in the $Assignment2.py$ file.

Exhaustive Search
Inside the *marsorbit.py* file, I have extracted the data once more to get the times and oppositions in the local scope of the file to ensure smooth running. I have defined many funcitons, but the main funciton is the $bestMarsOrbitParams$ function. Here I run an exhaustive search over all parameters except s, as we already have the approximate value for s which is 0.5241 degrees/days. I have used the *np.linspace* function from the numpy library for the exhaustive search over all the parameters.
I have seen that empirically the output converges much faster due to using the minimize funcition, hence I have set up a tolerance term, that if there is no improvement for 10,000 runs of the loop, I will exit out and return the values as the berst fit.

Minimize
With a certain set of parameters, I try to narrow down the range of the parameters in a finer manner using the *minimize* function from the *scipy.optimize* library. I pass an error minimizing function, the inital guesses of the parameters from the exhuastive search, and the method as 'powell'.

Minimize - Powell Method
Here the minimize function is used to find the minima of a scalar function using the following parameters: a loss function, an initial guess, and a method. There are other parameters as well like args, but I did not go in depth into them. I used the 'powell' method as it is a conjuagate direction method that doesn't require the gradient of the function. This is useful as the function I am trying to minimize might not be differentiable at all places.

Although, this comes with a tradeoff that it might take longer to converge than using other methods. The method iteratively searches along a set of directions to find the minimum. It does a line search along all these directions and after minimizing along all these directions, it updates the current point to the minimum of all these directions. It also updates all directions to maintain conjugancy.
For the line search, it uses golden line search to get find the minima of the function to be minimized in a certain direction. This is where we take an interval [a,b] and take two internal points calculated using the golden ratio. Evaluate the function at the two points and update the new interval to be checked based on the function values at the two points.
It repeats this until convergence criteria is met. This method has the obvious disadvantage that it might get stuck in local minima, but that is taken care of by the exhaustive search we are doing over all the variables.

Minimize - Function to be optimiized
The error function $MarsEquantOrbit$ takes in all the parameters and calculates the error between the predicted opposition longitudes and the actual opposition longitudes. We call this the descrepancy. I calculate the descrepancy for all the oppositions and take the sum of squares of all these errors. I return this as the error needing to be optimized in the *minimize* function. I used sum of squares of errors instead of max error or even RMSE as I have tried empirically all three and have gotten the best results with sum of squares.

Error Function
Now the minimize function outputs the list of parameters with the least error. We take these parameters and pass it through $Errors\_and\_MaxError$ function which calculates all the errors for the twelve oppositions and the

maximum error with the proper sign. I return this to and compare with the current maxerror, if the new error is less than the current error, I update the current error and the best parameters.

Helper Functions and Exception Handling
All this is in a try and except block as while calculating the descrepancies, I have to use two equations to find the intersection of the equant longitudes with the predicted orbit. Once we get this intersection we can calculate heliocentric longitudes of these intersection points and compare with opposition longitudes to get descrepancy.
To calculate this intersection point I use the functions $y\_intersection$ and $quad\_form$ to caluate the y-coordinate and x-coordinate respecitvely. Now while calculating this intersection point I use the quadratic formula to calculate the x coordinate. In some cases the discriminant may be negative, while will throw an error handled by the $raiseException$ function. There is also the case where the minimize funciton throws an error for a given set of parameters. This is all handled by the try and except block.

Output
I only update the parameters and max_error if the new error is less than 0.06 and less than the current max_error. This way I can get the best parameters and the minimum of the maximum errors over different sets of parameters.
Once we have the best parameters and the minimum of the maximum errors over different sets of parameters, I output this to the $Assignment2.py$ file and print it out.