ASEN 4057: Assignment #6

Jake Ursetta & Nikolas Setiawan
University of Colorado, Boulder

Introduction

Three astronaut on their way to the moon are surprised with a sudden a series of explosion. The life support is now compromised and the spacecraft is on a collision course with the moon.

In this assignment, a function written in the language c was used to calculate the change in velocity in order to return the astronauts safely to earth. However, the function was written in order to optimize two objectives. The first is the lowest change in velocity allowing the astronauts to return to earth. The second is the change in velocity ranging from 0 to 100 $\frac{m}{s}$ that would allow for the quickest return to earth.

Command Line Instruction

The threeBody program takes three command line arguments. The first is the objective, either 1 or 2. The second is the clearance around the moon in meters. The third is the absolute accuracy of the change in velocity. An an example of the call for objective 1, with a 10,000 meter clearance and an absolute accuracy of $.5\frac{m}{s}$ in the terminal is the following, "./threeBody 1 10000 .5".

The bash shell script which calls the c function only requires that the "threeBody" compiled file is in the same folder. It will then generate the requested output files in the folder "output".

The output files follow the correct labeling as described in the assignment description document. The rows correspond to each time step. the first column is the time in seconds. The next four columns correspond to the spacecrafts x position, y position, x velocity, and y velocity respectfully. The next four columns are the same information in the same order but for the moon. The final four columns are again the same information in the same order but for the earth.

Function Scope and Design Process

The code for the C function is based off of the MATLAB code created in Assignment 2. However, because C is a lower level language, the luxury of functions such as fminsearch or ode45 are not available. Instead, our group focused on splitting the main program into 3 functions.

Main Function Scope

The first function is the main script. The purpose of the main script is to allocate the initial conditions, generate changes of velocities and angles for the spacecrafts flight, call a function to integrate the flight path with the newly generated initial velocity, and determine the minimum ΔV required to return to earth and the ΔV for the minimum time to return to earth.

Main Function Design Process

The function was originally created to only check if the initial conditions resulted in a lunar crash. Once this was a success, a for loop was added in order to alter the initial velocity of the spacecraft. The for loop was designed to increase velocity from 0 to 100. This was done so that under objective 1, the first velocity found could be recorded as the optimum change in velocity. An if else state was then used to determine the objective given by the user. The else portion of the statement contained a similar for loop to the one above, however it allowed the for loops to run over the entire velocity and angle change in order to insure the minimum time to return was found. In order to speed up the integration, once a successful return was found, the time of flight was recorded. This time was passed to the integration function and if the integration function ran longer than the previous successful time, the integration was terminated. This allowed for a much short run time.

Integration function Scope

The second function is the Runge Kutta integration function. This function uses pointers passed by the main function to get initial conditions of the earth, moon, and spacecraft. This function then uses a while loop to iterate through time steps and integrate the path of the earth, moon, and spacecraft. The integration is done using the Runge Kutta method. In order to do this, The function passes the current states of the three bodies to a "derivs" function which then returns the derivatives of each state. The integration function uses the derivs function 4 times in order to generate 12 "k" values which are used in the linear integration. Given the assumption that the derivs function is "a", the following figure shows how the derivs function is used to create each k value.

$$
\begin{aligned}
\vec{k}_{1_{v_{i+1}}} &= \vec{a}\left(\vec{r_i}\right) \\
\vec{k}_{2_{v_{i+1}}} &= \vec{a}\left(\vec{r_i} + \vec{k}_{1_{r_{i+1}}}\frac{h}{2}\right) \\
\vec{k}_{3_{v_{i+1}}} &= \vec{a}\left(\vec{r_i} + \vec{k}_{2_{r_{i+1}}}\frac{h}{2}\right) \\
\vec{k}_{4_{v_{i+1}}} &= \vec{a}\left(\vec{r_i} + \vec{k}_{3_{r_{i+1}}}h\right)
\end{aligned}
$$

Figure 1: General Runge Kutta Method

It should be noted that these K values are actually each a vector that is 12 values in length. This is because that for the 3 bodies, a K value must be generated for the x and y position and velocity. This results in 4 K values per body. The integration function uses this integration method in order to predict the orbital path of all three bodies. The function is terminated when one of the following 3 conditions is met. First, the spacecraft returns to earth. This is checked for if the normal of the spacecrafts position vector is smaller than the radius of the earth. The second is if the spacecraft hits the moon. This is done similarly to the above check but the normal of the vector between the spacecraft and the moon and the moons radius are used. Finally, the last check is if the normal of spacecrafts position vector is twice that of the normal of the position vector of the moon. Once one of these conditions is met, the integration ends and if the spacecraft successfully landed back on earth, a 1 is returned to the main function to indicate the successful return.

Integration function Design Process

The function was originally created using the Euler method. However, it was determined that the small time step required for accuracy took too long to run with such a large number of trials. The Runge Kutta method was then used instead in order to allow for larger time steps. The Runge Kutta method required a new "derivs" function which returned the derivative of all 12 states for the three bodies. the function was designed to create the k vectors as a 12x4 matrix. This allowed for K1 through K4 of all 12 bodies to be stored. a for loop was used in order to generate each K1 through K4 value and reduce the total number of lines of code. The function was also later altered in order to speed up objective 2. This was done by allowing the function to be called with an additional variable "tmin". "tmin" was infinite until a successful landing was predicted. Once this successful landing was predicted "tmin" was the smallest time of return of the previous predictions. If the integration ran longer than this minimum time, the integration was terminated and the case was not considered a success. This insured the integration did not run longer than needed.

Derivative function scope

The final function is the derivative function mentioned above. This function takes in 3 state vectors each containing the position and velocities of the earth, moon, and spacecraft. The function than uses newtons law of gravitation and the known mass of each body to determine the acceleration and therefore the change in velocity. The function also uses the velocity in the state vectors to determine the new positions.

Derivative function Design Process

The function was initially created in order to handle the Euler method and simply returned the acceleration for just the moon and the spacecraft. However, once the integration function was changed to handle the Runge Kutta integration method, the derivative function was modified. The modification involved accepting the state vectors of the earth, moon, and spacecraft. The function then returned the derivative either based on the velocity or newtons law of gravitation. This allowed for a more streamlined approach as outlined in the integration function section.

Profile

Objective one

Table 1: Objective One Flat Profile

% Time	Cumulative seconds	Self Seconds	Calls	Self ns/call	Total ns/call	Name
61.05	1.00	1.00	93519736	10.71	10.71	deriv
38.46	1.63	0.63				rungeKutta.constprop.0
0.61	1.64	0.01				frame_dummy

From the table, we can see that the most time consuming function is deriv. This is due to the fact that deriv is called thousands of times over the course of one integration. Additionally, the integration call is also called a multitude of times for the various different changes in velocities and angles. This shows that if This function could be optimized and time could be decreased by even a small amount, The total time could be drastically reduced.

Table 2: Objective One Call Graph

Index	% Time	Self	Children	Called	Name
[1]	99.4	0.63	1.00	93519736/93519736	rungeKutta.constprop.0 [1]
		1.00	0.00		deriv [2]
[2]	61.0	1.00	0.00	93519736/93519736	rungeKutta.constprop.0 [1]
		1.00	0.00	93519736	deriv [2]
[3]	0.6	0.01	0.00		<spontaneous>
					frame_dummy [3]

From the Call Graph table above, it can be seen that the Runge Kutta integration function is run the majority of the time. However, its child function deriv is the culprit for almost twice of its own time. This indicates again that the deriv function should be optimized or the total number of calls should be lowered in order to see the greatest impact on time.

Objective two

Table 3: Objective Two Flat Profile

% Time	Cumulative seconds	Self Seconds	Calls	Self ns/call	Total ns/call	Name
55.92	3.49	3.49	325412436	10.72	10.72	deriv
43.48	6.20	2.71				rungeKutta.constprop.0
0.72	6.25	0.05				frame_dummy

Just like for objective one, the function deriv took the most time in the code. That's because the code is using the same calculation method for finding objective two's optimized change in velocity. However, it is taking longer because the function checks all changes in velocity over the $0\frac{m}{s}$ to 100 $\frac{m}{s}$ constraint.

Table 4: Objective Two Call Graph

Index	% Time	Self	Children	Called	Name
[1]	99.3	2.71	3.49	325412436/325412436	rungeKutta.constprop.0 [1]
		3.49	0.00		deriv [2]
[2]	55.8	3.49	0.00	325412436/325412436	rungeKutta.constprop.0 [1]
		3.49	0.00	325412436	deriv [2]
[3]	0.7	0.05	0.00		<spontaneous>
					frame_dummy [3]

The above table again indicates the same finding from the objective one profile. While the Runge Kutta function is responsible for the vast majority of run time, its child 'deriv' actually takes the majority of the entire functions call time. However, Because the main function is calling the Runge Kutta integration over the entire change in velocity constraint, there are more calls and the entire function takes much longer.

References

[1]Prof. Evans, *ASEN 4057 Assignment 6: C Software Design*, D2l.