

# ASEN 4057 - Assignment 2

Group 1: Nathan Herr\*, Bailey Topp†

February 6, 2019

*University of Colorado Boulder*

## I. Part 1

### A. Software Block Diagram

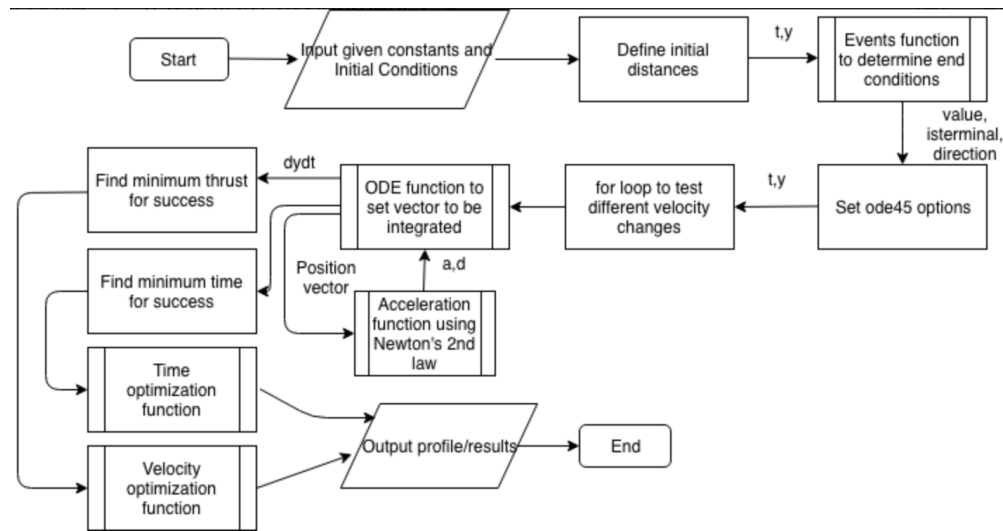


Figure 1: Software Block Diagram

Figure 1, shown above, is a software block diagram showing the organization and order of the code developed to accomplish Objectives 1 and 2.

### B. Objective #1

The minimum burn, or  $\|\Delta V\|$ , that the Apollo 13 capsule will be able to return to Earth with is  $\Delta V = (0.326, 49.105)$  [m/s] corresponding to an (x,y) format. This solution was obtained by first assigning success or failure indices to burns within a coarse grid of  $[-100:100]$  range in the both the x and y, then corresponding those indices to their (x,y) burns. The minimum of these burns that also corresponded with a successful return was found within the coarse grid, then used as an initial guess for the `fminsearch()` function in MATLAB. Figure 2 represents the trajectory of the spacecraft and Moon as this simulation played out.

---

\*SID: 106545930

†SID:104232674

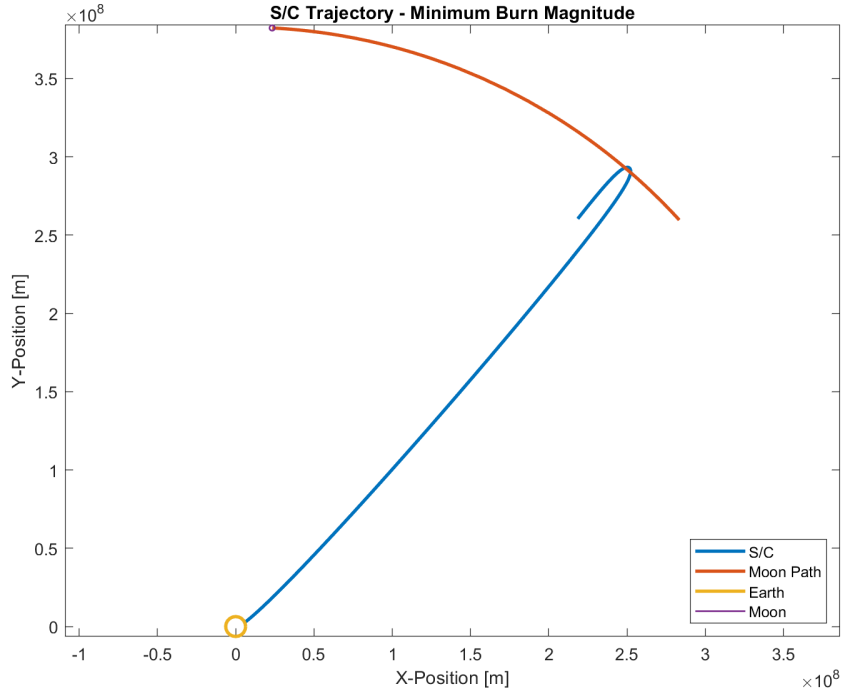


Figure 2: Minimum burn for return, spacecraft and Moon trajectories

### C. Objective #2

The burn that the Apollo 13 capsule will be able to return to Earth with a minimized time and  $|\Delta V| \leq 100$  [m/s] is  $\Delta V = (-0.000, 49.108)$  [m/s] corresponding to an (x,y) format. This returns the spacecraft to the Earth in about 290791 [s] or 80.775 [hrs]. This solution was obtained by first matching all successful burns with a magnitude of less than 100 [m/s] to a success or failure index, then corresponding those indices to a "time to end of simulation" output from the A2\_events.m function. The minimum of these times that also corresponded with a successful return was found using a coarse grid search, then using the solution of the coarse method as an initial guess for the fminsearch() function in MATLAB. Figure 3 represents the trajectory of the spacecraft and Moon as this simulation played out.

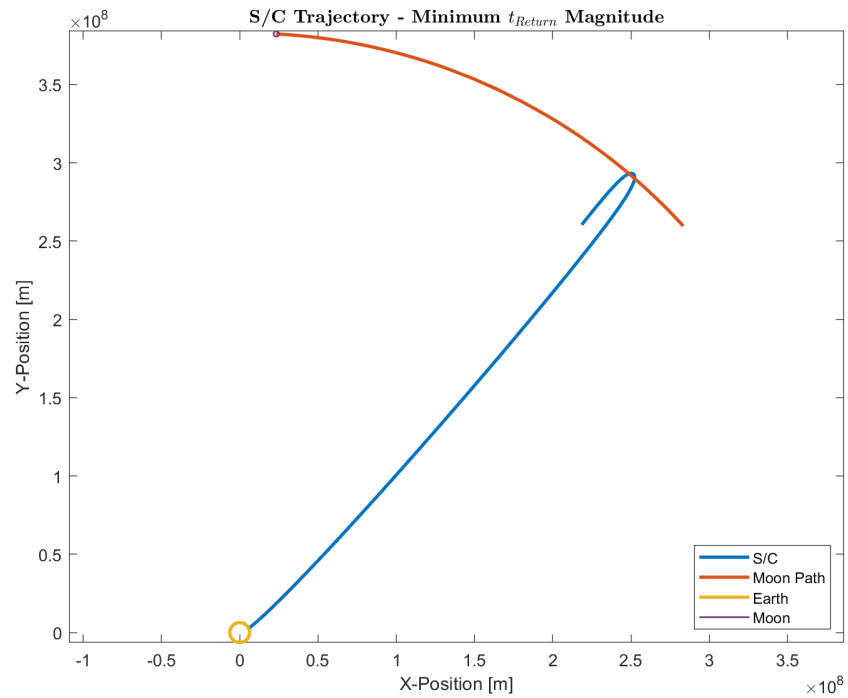


Figure 3: Minimum time of return, spacecraft and Moon trajectories

## II. Part 2

### A. Profile Summary

Below is the overall profile summary of the entire script and its notable functions for Assignment 2.

## Profile Summary

Generated 04-Feb-2019 10:30:09 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
<a href="#">A2_Main</a>	1	9.664 s	0.111 s	
<a href="#">ode45</a>	1056	9.160 s	1.735 s	
<a href="#">A2_ODE</a>	412062	4.906 s	2.013 s	
<a href="#">fminsearch</a>	2	4.281 s	0.042 s	
<a href="#">A2_Main&gt;@(t,y)A2_ODE(t,y)</a>	220941	3.050 s	0.407 s	
<a href="#">A2_Accel</a>	412062	2.893 s	2.893 s	
<a href="#">...gt:@(x_opt)A2_OptFunc_V(x_opt,y0_opt)</a>	156	2.439 s	0.001 s	
<a href="#">A2_OptFunc_V</a>	156	2.437 s	0.008 s	
<a href="#">...gt:@(x_opt)A2_OptFunc_T(x_opt,y0_opt)</a>	99	1.797 s	0.001 s	
<a href="#">A2_OptFunc_T</a>	99	1.796 s	0.006 s	
<a href="#">A2_OptFunc_V&gt;@(t,y)A2_ODE(t,y)</a>	116652	1.560 s	0.208 s	
<a href="#">A2_OptFunc_T&gt;@(t,y)A2_ODE(t,y)</a>	74469	1.053 s	0.142 s	
<a href="#">funfun\private\odezero</a>	51761	1.043 s	0.883 s	
<a href="#">funfun\private\intrp45split</a>	51761	0.442 s	0.442 s	
<a href="#">legend</a>	1	0.186 s	0.006 s	
<a href="#">legend&gt;make_legend</a>	1	0.180 s	0.008 s	
<a href="#">funfun\private\odearguments</a>	1056	0.138 s	0.046 s	
<a href="#">odeget</a>	11616	0.126 s	0.047 s	
<a href="#">A2_events</a>	60069	0.110 s	0.110 s	

Figure 4: The overall profile summary of Assignment 2

### B. Detailed Profile of A2\_ODE.m Function

The following detailed report shows the breakdown for A2\_ODE.m, the function which sets the equations of motion for ode45 and was the most time-consuming sub-routine.

## A2\_ODE (Calls: 412062, Time: 4.906 s)

Generated 04-Feb-2019 10:30:19 using performance time.

function in file

[C:\Users\bato9895\Downloads\drive-download-20190204T165207Z-001\A2\\_ODE.m](C:\Users\bato9895\Downloads\drive-download-20190204T165207Z-001\A2_ODE.m)

[Copy to new window for comparing multiple runs](#)






Refresh

- ☒ Show parent functions      ☒ Show busy lines      ☒ Show child functions  
☒ Show Code Analyzer results      ☒ Show file coverage      ☒ Show function listing


### Parents (calling functions)

Function Name	Function Type	Calls
<a href="#">A2_Main&gt;@(t,y)A2_ODE(t,y)</a>	anonymous function	220941
<a href="#">A2_OptFunc_V&gt;@(t,y)A2_ODE(t,y)</a>	anonymous function	116652
<a href="#">A2_OptFunc_T&gt;@(t,y)A2_ODE(t,y)</a>	anonymous function	74469

### Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
<a href="#">14</a>	[a,~] = A2_Accel(y(1:4));	412062	3.537 s	72.1%	
<a href="#">17</a>	dydt(5:8) = a;	412062	0.592 s	12.1%	
<a href="#">20</a>	dydt(1:4) = y(5:8);	412062	0.369 s	7.5%	
<a href="#">24</a>	end	412062	0.097 s	2.0%	
<a href="#">22</a>	dydt = dydt';	412062	0.013 s	0.3%	
All other lines			0.299 s	6.1%	
Totals			4.906 s	100%	

### Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
<a href="#">A2_Accel</a>	function	412062	2.893 s	59.0%	

Self time (built-ins, overhead, etc.)			2.013 s	41.0%	
Totals			4.906 s	100%	

### Code Analyzer results

Line number	Message
<a href="#">11</a>	Input argument 't' might be unused, although a later one is used. Consider replacing it by ~.

### Coverage results

[Show coverage for parent directory](#)

Total lines in function	14
Non-code lines (comments, blank lines)	9
Code lines (lines that can run)	5
Code lines that did run	5
Code lines that did not run	0
Coverage (did run/can run)	100.00 %

### Function listing

Color highlight code according to

time	Calls	line
		11 function [dydt] = A2_ODE(t,y)
		12
		13 % Call acceleration function
3.537	412062	<a href="#">14</a> [a,~] = <a href="#">A2_Accel</a> (y(1:4));
		15
		16 % Set velocity
0.592	412062	<a href="#">17</a> dydt(5:8) = a;
		18
		19 % Set position
0.369	412062	<a href="#">20</a> dydt(1:4) = y(5:8);
		21
0.013	412062	<a href="#">22</a> dydt = dydt';
		23
0.097	412062	<a href="#">24</a> end

### **C. Detailed Profile of A2\_Accel.m Function**

The following detailed report shows the breakdown for A2\_Accel.m, the function which defines the acceleration of the spacecraft at a given location in (x,y) space. This was the second-most time-consuming sub-routine.

## A2\_Accel (Calls: 412062, Time: 2.893 s)

Generated 04-Feb-2019 10:30:47 using performance time.

function in file

[C:\Users\bato9895\Downloads\drive-download-20190204T165207Z-001\A2\\_Accel.m](C:\Users\bato9895\Downloads\drive-download-20190204T165207Z-001\A2_Accel.m)

[Copy to new window for comparing multiple runs](#)







Refresh

- ☒ Show parent functions      ☒ Show busy lines      ☒ Show child functions  
☒ Show Code Analyzer results    ☒ Show file coverage    ☒ Show function listing

### Parents (calling functions)

Function Name	Function Type	Calls
<a href="#">A2_ODE</a>	function	412062

### Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
<a href="#">27</a>	d(1) = sqrt((p(1)-p(3))^2+(p(2...	412062	0.625 s	21.6%	
<a href="#">43</a>	a(1) = (FMSx + FESx)/mS;	412062	0.442 s	15.3%	
<a href="#">28</a>	d(2) = sqrt((p(1)-0)^2+(p(2)-0...	412062	0.387 s	13.4%	
<a href="#">44</a>	a(2) = (FMSy + FESy)/mS;	412062	0.333 s	11.5%	
<a href="#">48</a>	end	412062	0.102 s	3.5%	
All other lines			1.003 s	34.7%	
Totals			2.893 s	100%	

### Children (called functions)

No children

### Code Analyzer results

No Code Analyzer messages.



## Coverage results

[Show coverage for parent directory](#)

Total lines in function	37
Non-code lines (comments, blank lines)	19
Code lines (lines that can run)	18
Code lines that did run	18
Code lines that did not run	0
Coverage (did run/can run)	100.00 %

## Function listing

Color highlight code according to

time	Calls	line
		12 function [a,d] = A2_Accel(p)
		13
		14 %% Given
		15
		16 % Note: p(1) = xS, p(2) = yS, p(3) = xM, p(4) =
		17
		18 % Mass properties
0.016	412062	<u>19</u> mM = 7.34767309*10^22; % [kg]
0.014	412062	<u>20</u> mE = 5.97219*10^24; % [kg]
0.013	412062	<u>21</u> mS = 28833; % [kg]
0.013	412062	<u>22</u> G = 6.674*10^-11; % [N(m/kg)^2]
		23
		24 %% Solution
		25
		26 % Caclulate distance
0.625	412062	<u>27</u> d(1) = sqrt((p(1)-p(3))^2+(p(2)-p(4))^2); % [m]
0.387	412062	<u>28</u> d(2) = sqrt((p(1)-0)^2+(p(2)-0)^2); % [m] Dist
0.100	412062	<u>29</u> d(3) = sqrt((p(3)-0)^2+(p(4)-0)^2); % [m] Dist
		30 % d(1) = dMS; d(2) = dES; d(3) = dEM;
		31
		32 % Calculate forces: x-components
0.075	412062	<u>33</u> FMSx = G*mS*mM*(p(3)-p(1))/d(1)^3; FMSx = -FMSx
0.060	412062	<u>34</u> FESx = G*mE*mS*(0-p(1))/d(2)^3; % [N]
0.059	412062	<u>35</u> FEMx = G*mE*mM*(0-p(3))/d(3)^3; % [N]
		36
		37 % Calculate forces: y-components
0.059	412062	<u>38</u> FMSy = G*mS*mM*(p(4)-p(2))/d(1)^3; FMSy = -FMSy

```

0.060 412062 39 FESy = G*mE*mS*(0-p(2))/d(2)^3; % [N]
0.059 412062 40 FEMy = G*mE*mM*(0-p(4))/d(3)^3; % [N]
41
42 % Add forces for acceleration
0.442 412062 43 a(1) = (FMSx + FESx)/mS;
0.333 412062 44 a(2) = (FMSy + FESy)/mS;
0.074 412062 45 a(3) = (FEMx + FSMx)/mM;
0.066 412062 46 a(4) = (FEMy + FSMy)/mM;
47
0.102 412062 48 end

```

## **D. Most Important Profile Results**

The most important results from the Matlab profile is that the two most time consuming functions were the ODE function and the acceleration function. This is important because the ODE function was the backbone of the program, as most of the results required a call to that function, and the acceleration was a child function of ODE function, so it was called every time the ODE function was. As these two functions were used the most for the solving of objectives 1 and 2, it makes sense that they would be called the most and thus have the longest run time within the main script. The ODE function was the most time-consuming subroutine, but had a lower self time than the acceleration function, meaning that the most of the run time for the ODE function was spent calling and running the acceleration function.

## **E. Methods for Improving Time**

One way to improve the time it takes the program to run would be to remove the call to the acceleration within the ODE function. While it is helpful for organization and helped with the debugging process, a call to a child function within a parent function is more time consuming than it would be to perform the function's subroutine within the ODE function. This would remove the need for the computer to jump into a separate function each time the ODE function was called, and thus improve the run time. Additionally, removing the second output, d, from the acceleration function would improve the run time slightly by not making the function have to create two outputs each time it is called when only one was used.