

ASEN 4057 Assignment #2: Apollo 13 Return Optimization

Keith Covington* and Torfinn Johnsrud†

February 9th, 2018

University of Colorado Boulder, Boulder, CO, 80309, U.S.

I. The Problem

The problem is the three body gravitational problem formed by the Earth, Moon, and a spacecraft. For the purpose of this assignment, the earth is assumed to be massive enough that it does not experience any acceleration or movement as a result of the spacecraft or Moon. The spacecraft is given initial conditions, and then a high order integration scheme is used to plot the trajectory of the spacecraft assuming that the gravitational forces from the Moon and Earth are the only forces acting on the spacecraft. With the given initial conditions, the spacecraft will crash into the moon. Given specific delta V's from the initial conditions, the spacecraft can return to Earth after using the Moon's gravitational assist. This assignment optimizes the optimal thrust and time that is required to return the spacecraft to Earth.

II. Solution Approach

First, the main script introduces the initial conditions of the spacecraft-Earth-Moon system. It then tries to optimize the initial conditions to minimize the magnitude of thrust needed to return the spacecraft to Earth. This is done by applying a coarse grid method of narrowing down an initial guess for optimization. The grid is composed of x- and y-components of vectors that span the realm of possible vectors at a moderate resolution. The program tries to apply the `fminsearch` minimization function at each of these locations in the 2-D space. It finds the minimum magnitude of velocity of all of these minimizations and then narrows the search area on the 2-D space. It again, at a finer resolution, applies the `fminsearch` minimization function at all of the grid points of the finer grid. The minimum of these optimization results is considered to be the optimal solution, as it has minimized from a sufficiently small grid. The function then uses these initial conditions to perform the numerical integration with `ode45`.

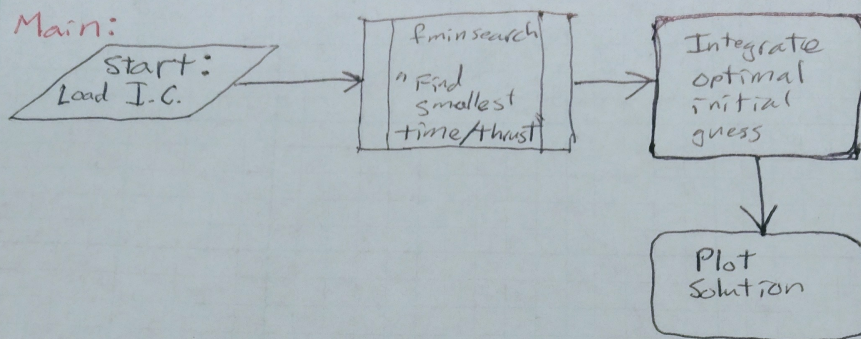
It should be noted that a more efficient approach could be taken by directly evaluating `ode45` at each of the grid points rather than running the `fminsearch` at each location. This would likely result in a significant decrease in run time.

Note that in the program submitted as is, it uses `ode45`. However, 4th order Runge-Kutta method can also be implemented with the RK4 code attached.

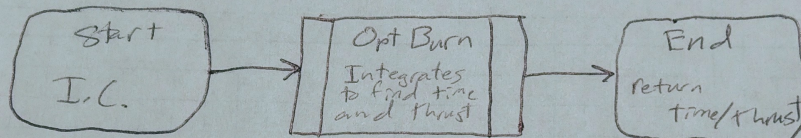
*SID: 105615600

†SID: 104424713

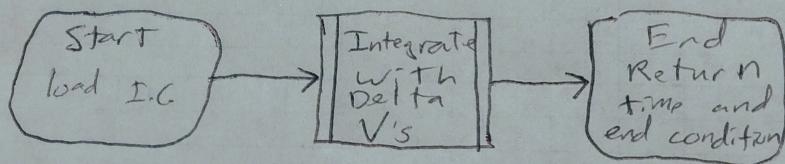
Main:



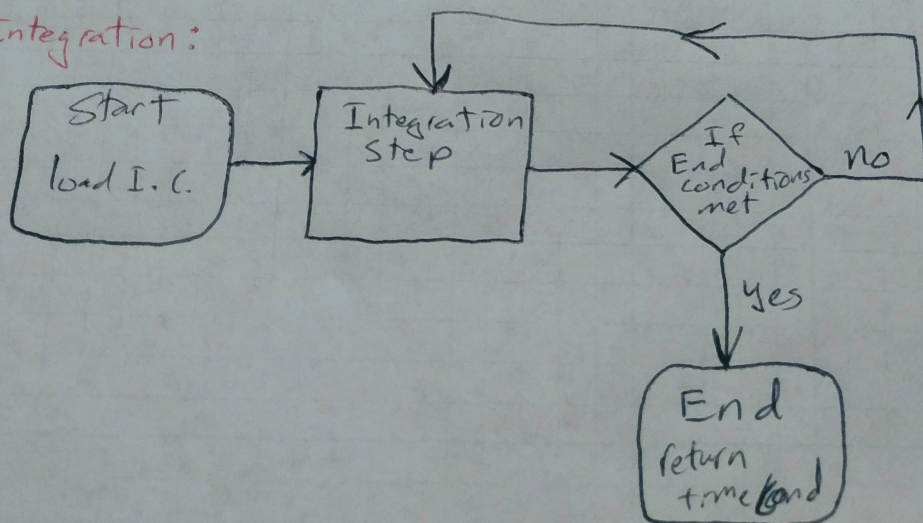
Fminsearch:



Opt. Burn:

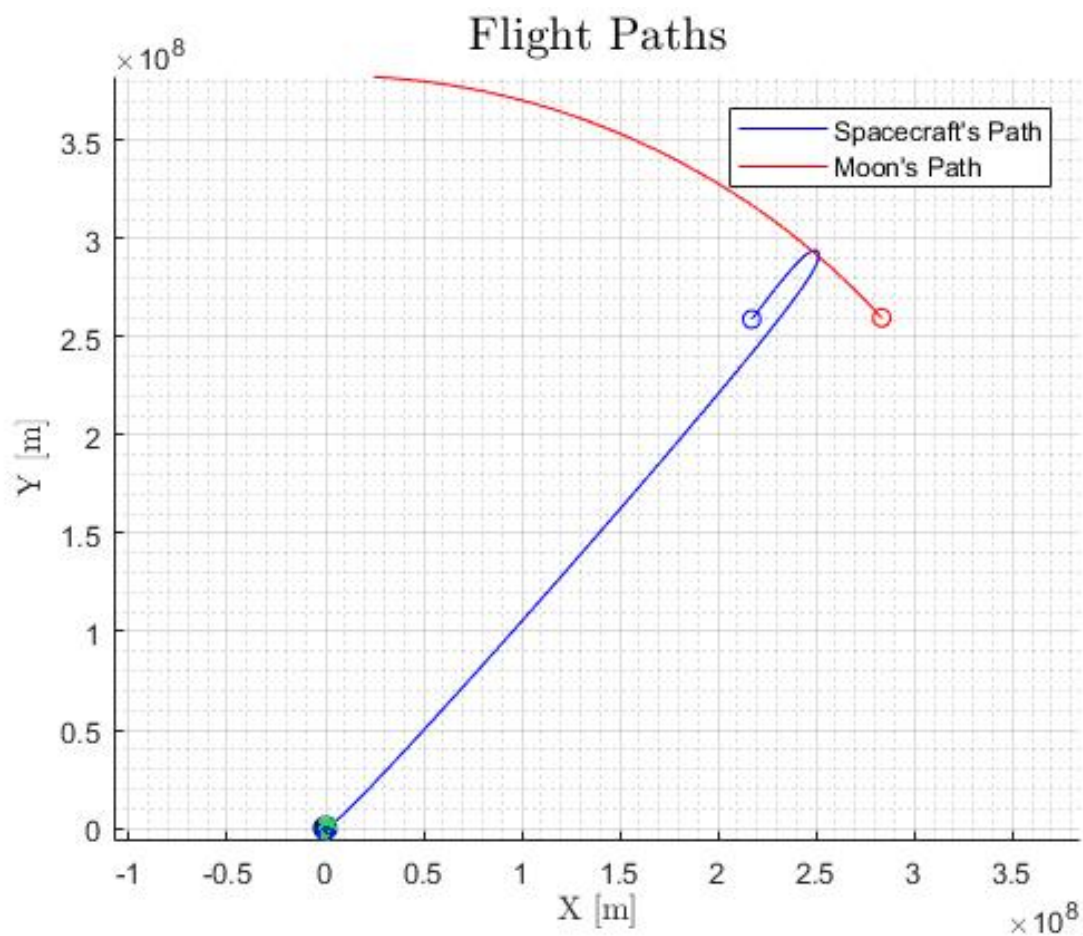


Integration:

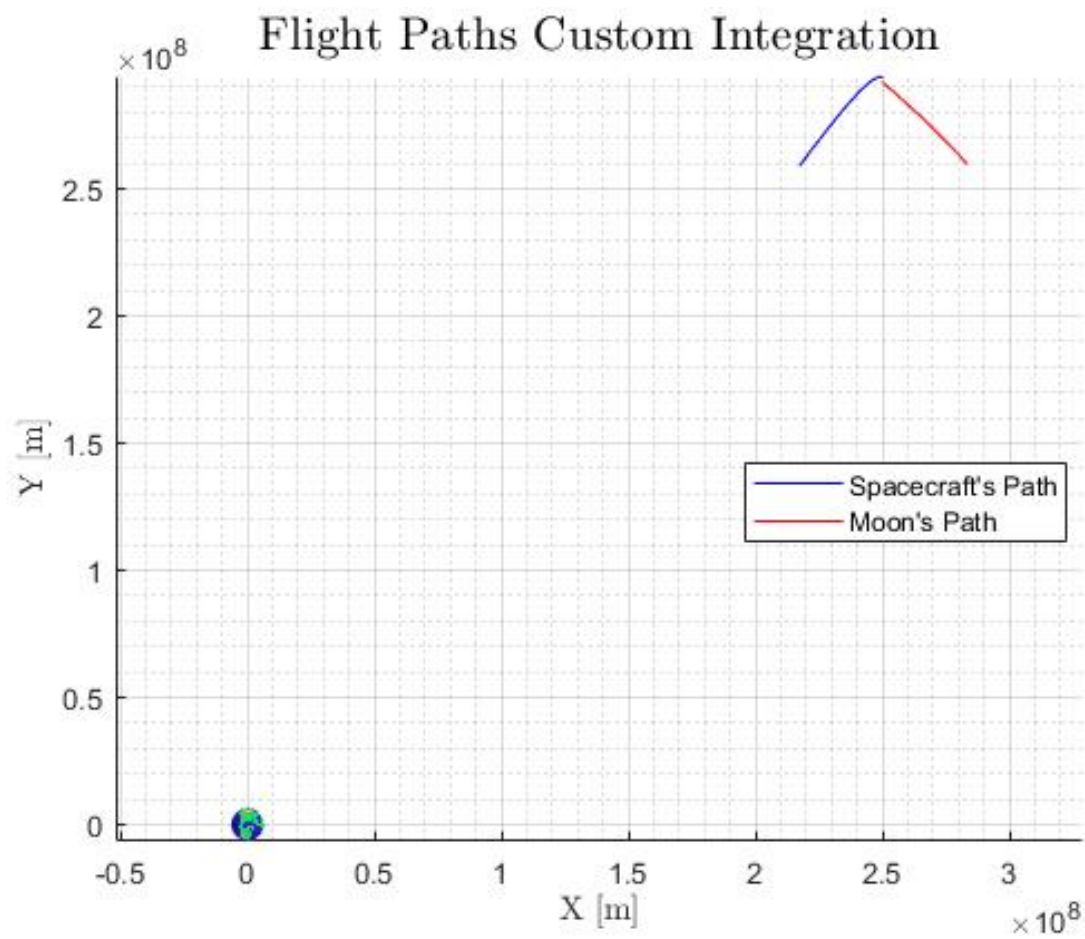


III. Results

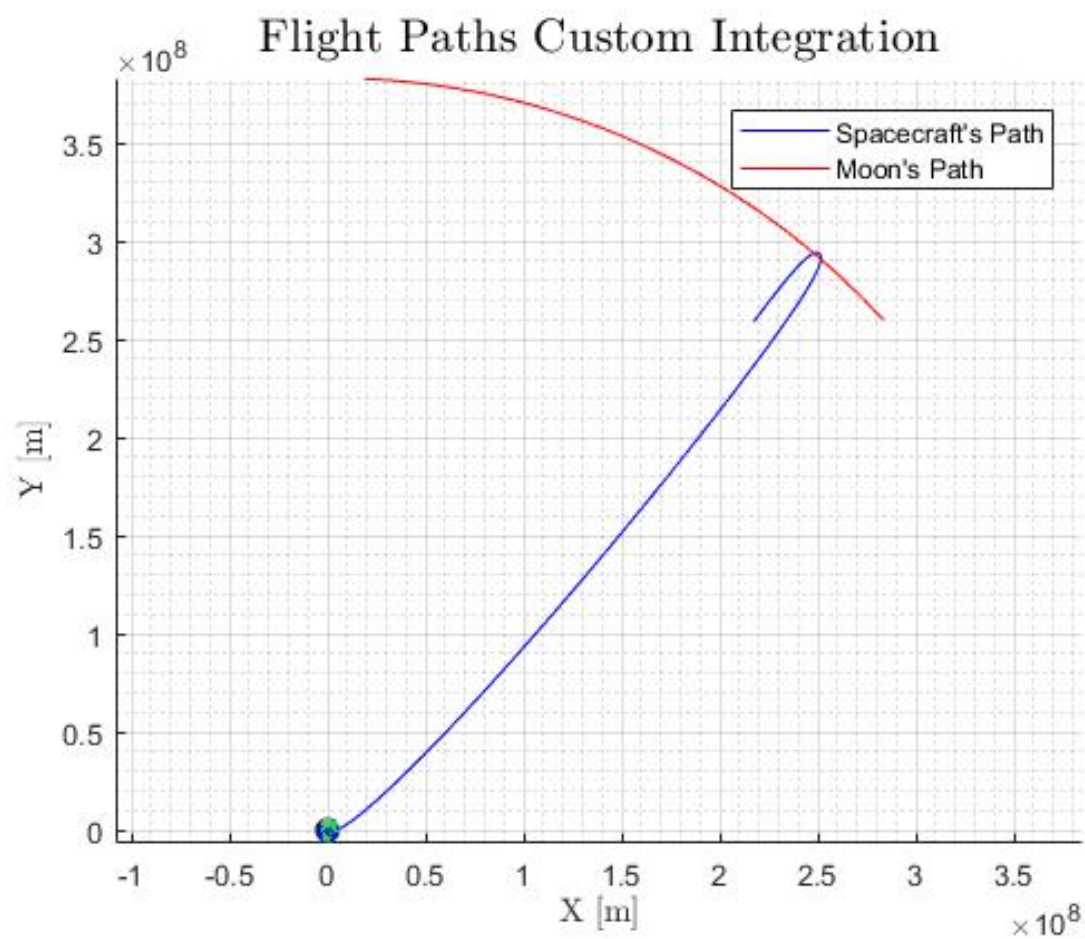
Optimal Delta V of $[-6.544565, 77.213700]$ m/s; The ODE45 result is as follows.



Using the custom integration program, the following result is found with the same initial conditions as above.



Due to integration differences, the satellite collides with the Moon. By varying the initial conditions minimally by adding $[0.6, 0.5] \text{ m/s}$, the following result is obtained. This small perturbation is enough to avoid collision with the moon and return the spacecraft to Earth.



IV. Program Performance

Main profile of the program

Profile Summary

Generated 10-Feb-2018 04:46:13 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Main	1	433.017 s	0.230 s	
GridOpt	1	431.949 s	0.504 s	
...param2].t.state.parameters,'thrust')	40400	431.435 s	0.298 s	
OptBurn	40400	431.137 s	1.846 s	
ode45	40401	424.200 s	122.974 s	
...state)ODEroutine(t.state.parameters)	11628698	206.707 s	48.838 s	
ODEroutine	11628951	157.873 s	114.377 s	
funfun\private\odezero	1527697	57.987 s	32.829 s	
Accel	11629775	43.498 s	43.498 s	
funfun\private\intrp45	1853600	29.170 s	29.170 s	
stopping_point	1853680	22.606 s	22.606 s	
funfun\private\odearguments	40401	5.252 s	1.829 s	
odeset	40403	5.113 s	4.501 s	
odeget	444411	4.953 s	2.064 s	
odeget>getknownfield	444411	2.889 s	2.889 s	
funfun\private\odefinalize	40401	2.332 s	2.332 s	
funfun\private\odeevents	40401	1.504 s	0.496 s	
funfun\private\odemass	40401	0.787 s	0.297 s	
strmatch	40403	0.612 s	0.612 s	
legend	2	0.608 s	0.007 s	
legend>make_legend	2	0.600 s	0.011 s	
Legend.Legend>Legend.Legend	2	0.271 s	0.022 s	
Legend.doSetup	2	0.208 s	0.024 s	
Legend.doMethod	4	0.149 s	0.002 s	
Legend.doMethod>set_contextmenu	2	0.146 s	0.036 s	
legend>find_legend	2	0.134 s	0.134 s	
Legend.Legend>Legend.set.Axes	2	0.120 s	0.001 s	
Legend.Legend>Legend.set.Axes_1	2	0.119 s	0.001 s	

The ODE45 performance

ode45 (Calls: 40401, Time: 424.200 s)

Generated 10-Feb-2018 04:48:20 using performance time.

function in file <C:\Program Files\MATLAB\R2017a\toolbox\matlab\funfun\ode45.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
OptBurn	function	40400
Main	script	1

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
353	odezero (@ntrp45,eventFcn,event...	1527697	70.155 s	16.5%	
261	f(:,2) = feval(odeFcn,t+hA(1),...	1931425	53.589 s	12.6%	
262	f(:,3) = feval(odeFcn,t+hA(2),...	1931425	46.688 s	11.0%	
263	f(:,4) = feval(odeFcn,t+hA(3),...	1931425	45.417 s	10.7%	
264	f(:,5) = feval(odeFcn,t+hA(4),...	1931425	44.728 s	10.5%	
All other lines			163.624 s	38.6%	
Totals			424.200 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
...state)ODEroutine(t,state.parameters)	anonymous function	11588298	205.669 s	48.5%	
funfun\private\odezero	function	1527697	57.987 s	13.7%	
funfun\private\ntrp45	function	1568018	26.050 s	6.1%	
funfun\private\odearguments	function	40401	5.252 s	1.2%	
funfun\private\odefinalize	function	40401	2.332 s	0.5%	
odeget	function	161604	1.637 s	0.4%	
funfun\private\odeevents	function	40401	1.504 s	0.4%	
funfun\private\odemass	function	40401	0.787 s	0.2%	
...state)ODEroutine(t,state.parameters)	anonymous function	252	0.006 s	0.0%	
Self time (built-ins, overhead, etc.)			122.974 s	29.0%	

Code Analyzer results

Line number	Message
356	The variable 'teout' appears to change size on every loop iteration. Consider preallocating for speed.
357	The variable 'yeout' appears to change size on every loop iteration. Consider preallocating for speed.
358	The variable 'ieout' appears to change size on every loop iteration. Consider preallocating for speed.
378	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.
379	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.
382	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.
383	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.
384	The variable 'f3d' appears to change size on every loop iteration. Consider preallocating for speed.
406	The variable 'tout_new' appears to change size on every loop iteration. Consider preallocating for speed.
407	The variable 'yout_new' appears to change size on every loop iteration. Consider preallocating for speed.
412	The variable 'tout_new' appears to change size on every loop iteration. Consider preallocating for speed.
414	The variable 'yout_new' appears to change size on every loop iteration. Consider preallocating for speed.
416	The variable 'yout_new' appears to change size on every loop iteration. Consider preallocating for speed.
427	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.
428	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.
431	The variable 'tout' appears to change size on every loop iteration. Consider preallocating for speed.
432	The variable 'yout' appears to change size on every loop iteration. Consider preallocating for speed.

Coverage results[Show coverage for parent directory](#)

Total lines in function	482
Non-code lines (comments, blank lines)	154
Code lines (lines that can run)	328
Code lines that did run	200
Code lines that did not run	128
Coverage (did run/can run)	60.98 %

Routine performance.

ODEroutine (Calls: 11628951, Time: 157.873 s)

Generated 10-Feb-2018 04:50:52 using performance time.

function in file [D:\Assignment2.2\drive-download-20180210T035359Z-001\ODEroutine.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
....state)ODEroutine(t.state.parameters)	anonymous function	11628698
....state)ODEroutine(t.state.parameters)	anonymous function	253

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
22	[aSx,aSy,aMx,aMy] = Accel(stat...	11628951	64.066 s	40.6%	
23	dydt(1) = state(3);	11628951	25.320 s	16.0%	
33	end	11628951	14.789 s	9.4%	
24	dydt(2) = state(4);	11628951	14.256 s	9.0%	
25	dydt(3) = aMx;	11628951	3.988 s	2.5%	
All other lines			35.454 s	22.5%	
Totals			157.873 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
Accel	function	11628951	43.495 s	27.6%	
Self time (built-ins, overhead, etc.)			114.377 s	72.4%	
Totals			157.873 s	100%	

Code Analyzer results

Line number	Message
1	Input argument 't' might be unused, although a later one is used. Consider replacing it by ~.
8	The value assigned to variable 'G' might be unused.
9	The value assigned to variable 'r_M' might be unused.
10	The value assigned to variable 'r_E' might be unused.