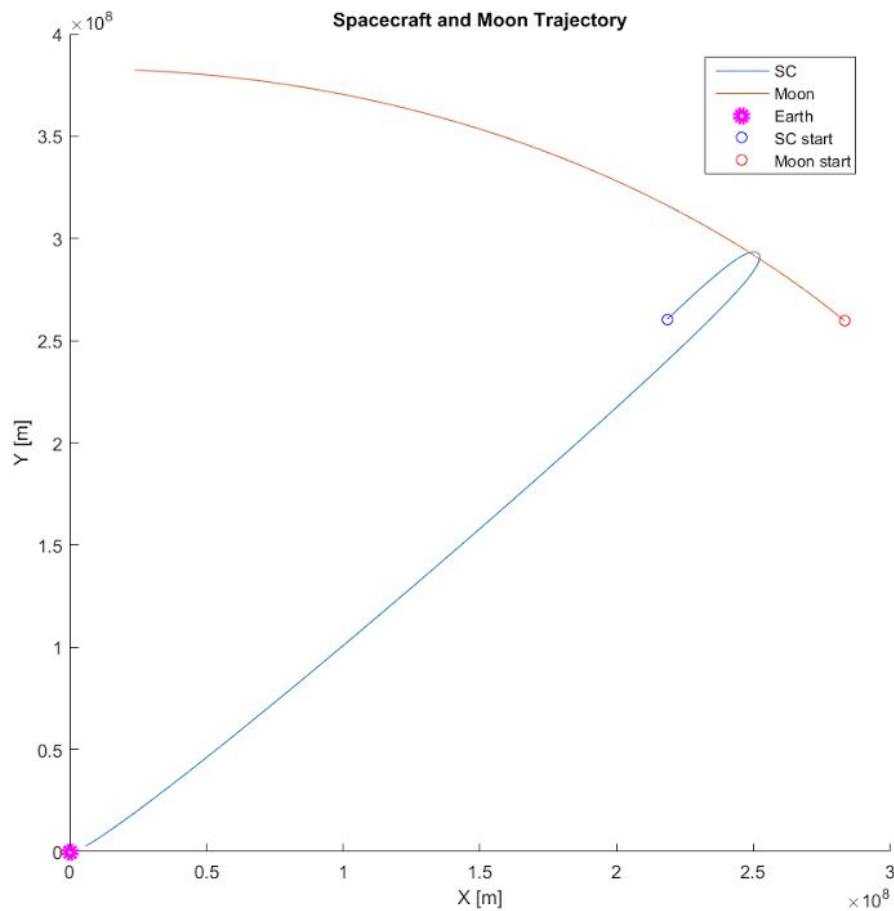


Part 1

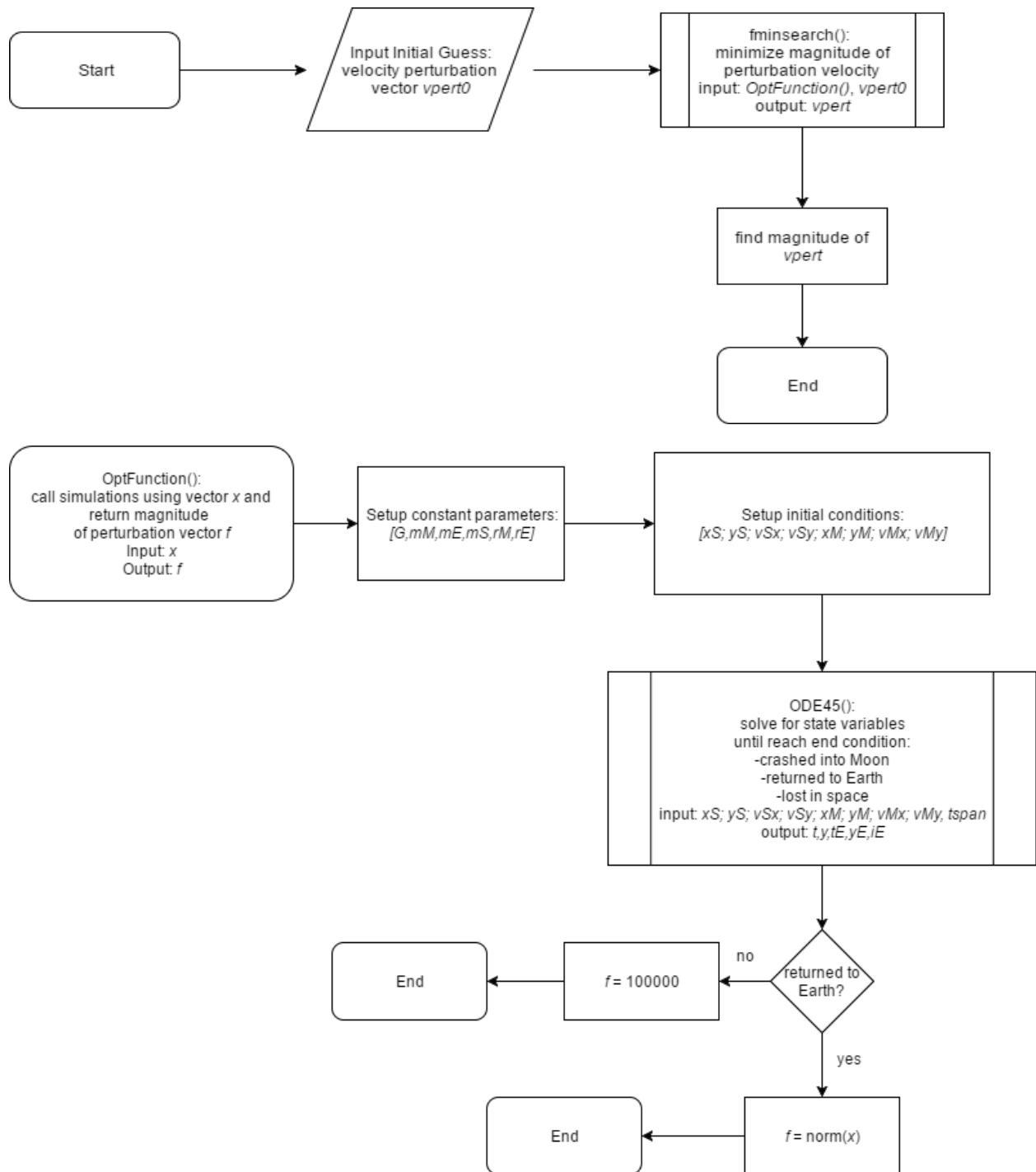
Final results for Objective 1:

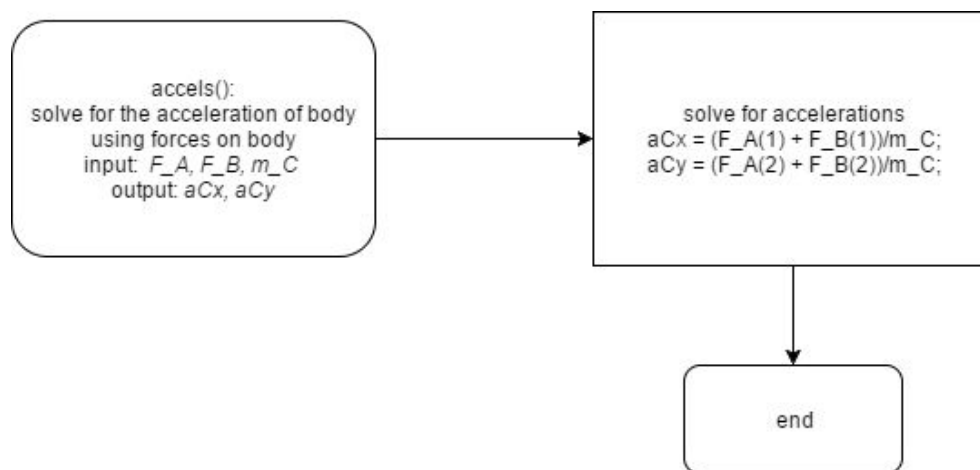
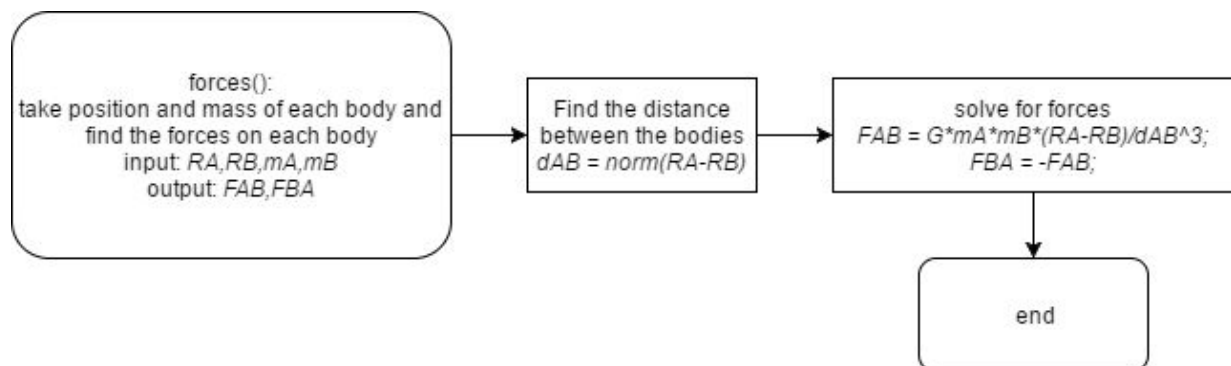
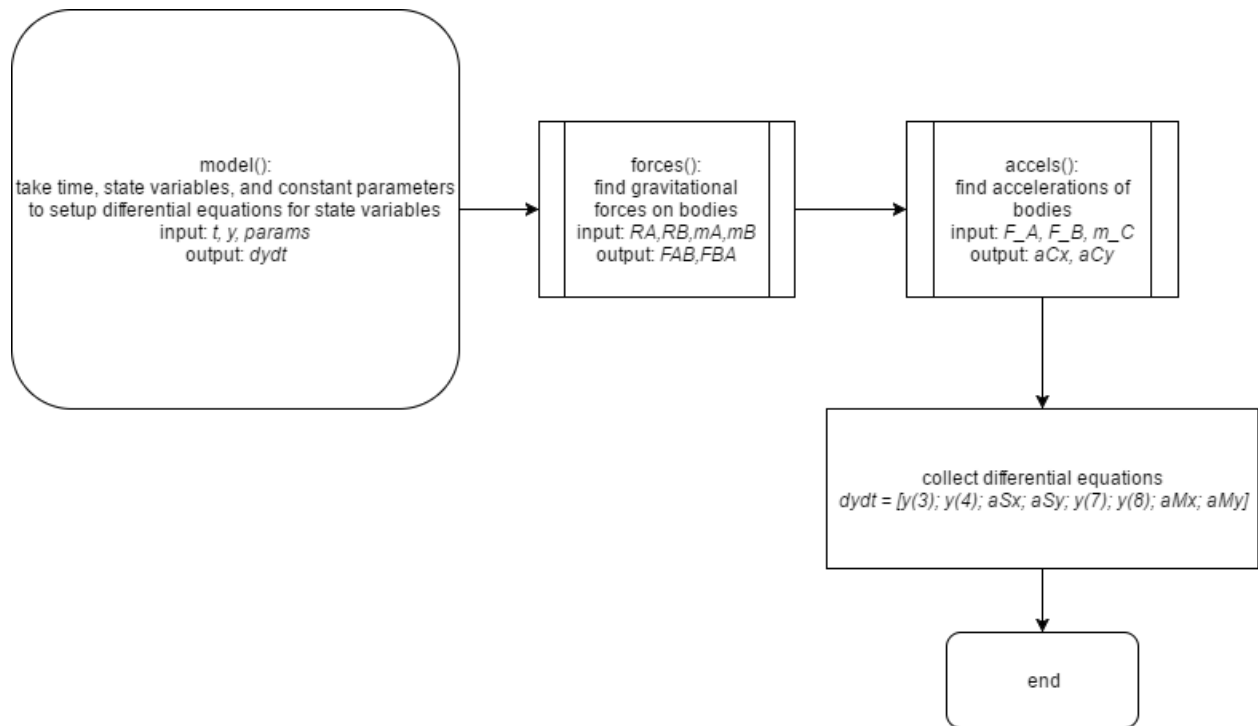
$v_{\text{pert}} = [0.3891, 49.0980] \text{ m/s}$

$v_{\text{pertmagnitude}} = 49.099514781123100 \text{ m/s}$



Below are the flowcharts that map the overall program control flow and control flow of individual subroutines

















Design Process

- Problem: Apollo 13 astronauts are on a crash course towards the moon. We know their position and initial velocity. The spacecraft only has a limited quantity of fuel left to perturb their velocity by a maximum magnitude of 100 m/s. Find the optimal thrust the astronauts should implement that will get them back home safely.
- Modular Program Design: Create basic functions to find forces and accelerations. Create a function that will keep track if the astronauts reached Earth, crashed, or became lost in space.
- Module Compositions: Create a function that combines modules to find rates of change of velocity and position.
- Module/Program Evaluation and Testing: Create a function that returns the velocity perturbation for the end conditions (crash, returned, lost, etc.) and test for optimization. Analyze Matlab profiler for improvements in code efficiency.
- Program Documentation: Comment each module

Part 2

Profile Summary

Generated 10-Feb-2017 16:20:51 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main2	1	10.119 s	0.005 s	
fminsearch	1	10.113 s	0.027 s	
main2>@(x)OptFunction(x)	101	10.083 s	0.005 s	
OptFunction	101	10.078 s	0.053 s	
ode45	101	9.994 s	2.559 s	
OptFunction>@(t,y)model(t,y,params)	265733	5.889 s	0.618 s	
model	265733	5.271 s	2.270 s	
forces	797199	2.386 s	2.386 s	
funfun\private\odezero	44272	0.873 s	0.648 s	
funfun\private\ntpr45	45227	0.624 s	0.624 s	
accels	531466	0.615 s	0.615 s	
spacecraft_stop	45227	0.216 s	0.216 s	
funfun\private\odearguments	101	0.037 s	0.019 s	
odeset	101	0.031 s	0.023 s	
odeget	1111	0.019 s	0.009 s	
funfun\private\odefinalize	101	0.012 s	0.012 s	
odeget>getknownfield	1111	0.010 s	0.010 s	
strmatch	202	0.008 s	0.008 s	
funfun\private\odeevents	101	0.007 s	0.005 s	
funfun\private\odemass	101	0.004 s	0.003 s	
optimget	8	0.003 s	0.003 s	
optimget>optimgetfast	8	0.001 s	0.001 s	
fcnchk	1	0.000 s	0.000 s	

model (Calls: 169803, Time: 3.496 s)

Generated 10-Feb-2017 17:00:28 using performance time.

function in file [C:\Users\mast9620\Documents\MATLAB\drive-download-20170210T230654Z\model.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
OptFunction>@(t,y)model(t,y,params)	anonymous function	169803

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
16	[FSM,FMS] = forces(RS,RM,mS...	169803	0.832 s	23.8%	
17	[FSE,FES] = forces(RS,RE,mS...	169803	0.691 s	19.8%	
18	[FME,FEM] = forces(RM,RE,mM...	169803	0.670 s	19.2%	
20	[aSx, aSy] = accels(FMS, FE...	169803	0.396 s	11.3%	
21	[aMx, aMy] = accels(FSM, FE...	169803	0.354 s	10.1%	
All other lines			0.553 s	15.8%	
Totals			3.496 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
forces	function	509409	1.599 s	45.7%	
accels	function	339606	0.406 s	11.6%	
Self time (built-ins, overhead, etc.)			1.492 s	42.7%	
Totals			3.496 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 ~.
5	The value assigned to variable 'G' might be unused.
9	The value assigned to variable 'rM' might be unused.
10	The value assigned to variable 'rE' might be unused.
17	The value assigned here to 'FSE' appears to be unused. Consider replacing it by ~.
18	The value assigned here to 'FME' appears to be unused. Consider replacing it by ~.

Coverage results

[Show coverage for parent directory](#)

Total lines in function	31
Non-code lines (comments, blank lines)	8
Code lines (lines that can run)	23
Code lines that did run	23
Code lines that did not run	0
Coverage (did run/can run)	100.00 %

Function listing

Color highlight code according to time ▼

time	Calls	line
		1 function dydt = model(t,y,params)
		2 %MODEL Summary of this function goes here
		3 % Detailed explanation goes here
		4 %% takes parameters out
< 0.01	169803	5 G = params(1); %N(m/kg)^2:
< 0.01	169803	6 mM = params(2);%kg
< 0.01	169803	7 mE = params(3);%kg
< 0.01	169803	8 mS = params(4);%kg
< 0.01	169803	9 rM = params(5);%m
< 0.01	169803	10 rE = params(6);%m
		11 %% pull out position of SC and Moon
0.05	169803	12 RS = [y(1),y(2)];
0.03	169803	13 RM = [y(5),y(6)];
< 0.01	169803	14 RE = [0,0];
		15 %% find forces on SC and Moon
0.83	169803	16 [FSM,FMS] = forces(RS,RM,mS,mM);
0.69	169803	17 [FSE,FES] = forces(RS,RE,mS,mE);
0.67	169803	18 [FME,FEM] = forces(RM,RE,mM,mE);
		19 %% find accels of SC and Moon
0.40	169803	20 [aSx, aSy] = accels(FMS, FES, mS);
0.35	169803	21 [aMx, aMy] = accels(FSM, FEM, mM);
		22 %% set up differential equations(velocities and accels)
0.09	169803	23 dydt = [y(3);...
	169803	24 y(4);...
	169803	25 aSx;...
	169803	26 aSy;...
	169803	27 y(7);...
	169803	28 y(8);...
	169803	29 aMx;...
	169803	30 aMy];
0.16	169803	31 end

forces (Calls: 509409, Time: 1.577 s)

Generated 10-Feb-2017 16:39:05 using performance time.
function in file C:\Users\mast9620\Documents\MATLAB\drive-download-20170210T230654Z\forces.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
model	function	509409

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
6	dAB = norm(RA-RB);	509409	0.557 s	35.3%	<div></div>
7	FAB = G*mA*mB*(RA-RB)/dAB^3;	509409	0.263 s	16.7%	<div></div>
9	end	509409	0.133 s	8.4%	<div></div>
8	FBA = -FAB;	509409	0.116 s	7.4%	<div></div>
5	G = 6.674E-11; %N	509409	0.018 s	1.1%	<div></div>
All other lines			0.490 s	31.1%	<div></div>
Totals			1.577 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	9
Non-code lines (comments, blank lines)	4
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

time	Calls	line
		1 function [FAB,FBA] = forces(RA,RB,mA,mB)
		2 %FORCES Summary of this function goes here
		3 % takes in the position vectors of 2 bodies and find the gravitational
		4 % force vectors between them
0.02	509409	5 G = 6.674E-11; %N
0.56	509409	6 dAB = norm(RA-RB);
0.26	509409	7 FAB = G*mA*mB*(RA-RB)/dAB^3;
0.18	509409	8 FBA = -FAB;
		9 end

The Matlab profiler is essentially equivalent to the “tic-tok” function, except the profiler provides much more detailed information about the time spent running the code, each individual function, each child function, and even down to every line of code. Knowing the execution time of each individual line in every function can assist the author in making code more efficient. We believe that the most important aspect of the profiler is the detailed analysis of how much time was spent on each individual function. Whether it be the calling function or a child function, this small, yet great tool can help pick out where your code is spending most of its time throughout execution. From there, one can find a more detailed analysis on how to improve each individual line that may or may not take plenty of time to run, but in general, Matlab essentially has the capability to tell the coder, “Hey, this function is taking a while to perform, maybe you can try improving it here.”

For our case, Matlab’s function ode45 spent the majority of the time throughout execution, however for the sake of simplicity, we will not be providing suggestions on how to improve a complicated Matlab function (some suggestions can already be found within the profiler.) In our code, the functions named “model” and “forces” were the next functions that took the majority of the run time. Reasons on why they run slow could be a multitude of answers, however, some answers could come from the profiler itself. For example, in the “model” function, the function is spending the most time on running “forces” and “accels”. One possible explanation is that these functions are called lots of times when running “model” once. A suggestion to improve this would be to call the functions once and also input how many different forces you would like to find such that each function doesn’t have to be called and ended multiple times back to back. Another suggestion would be to search for a built in Matlab function that already calculates gravitational forces between two objects in space. If one were looking for maximum speed, they could actually create code into another language that would need to be compiled rather than interpreted (although compiled would run much quicker, this would be much more difficult because there would be none of the built in functions that Matlab already provides for the writer.) Within the profiler for “model”, the code analyzer already presents us with some suggestions. There are multiple lines in which variables are unused, therefore looking at how long it takes to run each of these lines, then multiplying that times however many times “model” was called (169803 calls), this could add up to significant performance improvements for time. Next, inside the forces function, the majority of the time is actually spent on a line that already utilizes Matlab’s built in function for the magnitude of a vector. Since we can assume Matlab’s function are already better, more efficient code than we can make for now, the only suggestion we could make is to find another function, or since this specific line is subtracting 2 vectors, maybe the vectors could be input as already subtracted from each other such that this line would not have to compute two operations along one line. Again, it is

always possible that Matlab may already have a built in function for many of the functions already created in our script, which is why Matlab is an extremely efficient, simple, useful tool for software development and testing before taking it onto a compiler.