Apollo 13: Geospatial Software Design in Matlab

Justin Alvey [*], Azalee Rafii [†]

University of Colorado Boulder, Boulder, Colorado, 80310

February 11, 2017

I. Introduction

Matlab is a programming tool useful for low-scale software design and algorithm testing. Due to its nature as an interpretive programming language, Matlab is an intuitive language to learn than C++ or Fortran, which require compilation steps by the user. However, Matlab is already widely used in academia and for experimental testing through its plethora of useful software tools. Matlab, also known as Matrix Laboratory as its name suggests, is an excellent tool for solving complex linear systems.

Nomenclature

d_{ES0}	initial distance between the Earth and the Satellite (m)
v_{s0}	initial velocity of the satellite (m/s)
θ_s	Initial angle of the satellite (degrees)
d_{EM0}	initial distance between the Earth and the Moon (m)
θ_M	angle of the moon (degrees)
x_{E0}	initial x position of the Earth (m)
y_{E0}	initial y position of the Earth (m)
v_{Ex0}	initial x velocity of the Earth (m/s)
v_{Ey0}	initial y velocity of the Earth (m/s)
G	Gravitational Constant ($N(m/kg)^2$)
m_M	Mass of the Moon (kg)
m_E	Mass of the Earth
m_S	Mass of the spacecraft
r_M	Radius of the Moon (m)
r_E	Radius of the Earth
$xs0$	initial x position of the spacecraft (m)
$ys0$	initial y position of the spacecraft (m)
vx_{s0}	initial x velocity of the spacecraft(m/s)
vy_{s0}	initial y position of the spacecraft (m/s)

[*]103265867
[†]101715130

II. Part 1 - Matlab Software Design

A. Software Design Process and Flow Charts

The problem at hand involves solving a complex set of linear equations for modeling the motion of a spacecraft in Earth's orbit. Through software modeling in Matlab, the goal is to bring the satellite home safely, and avoid colliding with the moon. In order to complete this objective, a delta V must be applied to the spacecraft to change it's current trajectory. Since the spacecraft has limited fuel on board, the objective is to calculate the minimum delta V required.

Creating modular programs is a crucial design step for developing accurate software solutions. Since astronauts are currently on route to collide with the moon, the program must quickly and accurately make the necessary calculations. Figure (1) shows the main script to make these calculations.

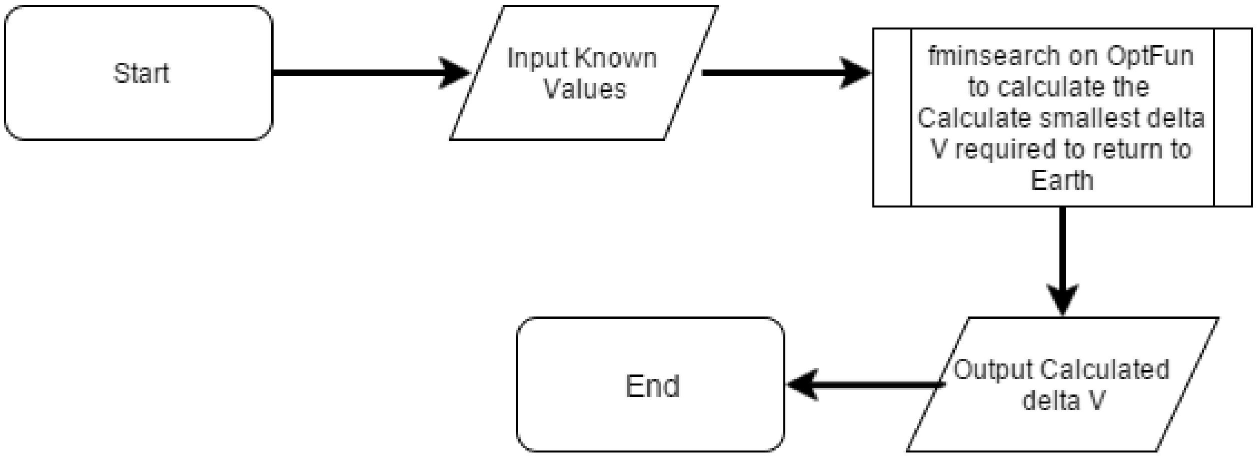

Start
Input Known Values
fminsearch on OptFun to calculate the Calculate smallest delta V required to return to Earth
Output Calculated delta V
End

Figure 1. Main Script Flow Chart

The logic of this flow chart is as follows. The known values are properties of the planets involved, as well as the current position and velocity of the spacecraft, moon, and earth: d_{ES0}, v_{s0}, θ_s, d_{EM0}, θ_M, x_{E0}, y_{E0}, v_{Ex0}, v_{Ey0}, G, m_M, m_E, m_S, r_M, r_E . From the known quantities, the other initial conditions can be calculated from equations (1 - 9).

$$x_{S0} = d_{ES} cos(\theta_S) \tag{1}$$

$$y_{S0} = d_{ES} sin(\theta_S) \tag{2}$$

$$v_{Sx0} = v_{S0} cos(\theta_S) \tag{3}$$

$$v_{Sy0} = v_{S0} sin(\theta_S) \tag{4}$$

$$x_{M0} = d_{EM0} cos(\theta_M) \tag{5}$$

$$y_{M0} = d_{EM0} sin(\theta_M) \tag{6}$$

$$v_{Mx0} = -v_{M0} sin(\theta_M) \tag{7}$$

$$v_{My0} = v_{M0} cos(\theta_M) \tag{8}$$

$$v_{M0} = \sqrt{\frac{G m_E^2}{(m_E + m_M) d_{EM0}}} \tag{9}$$

Once the initial conditions were calculated, a Matlab optimizing function needs to be called to calculate the minimum delta V required to return to Earth. Finally, the main function outputs the result of the calculation, and ends. The flow chart describing the function that is minimized by fminsearch is shown in Figure (2).

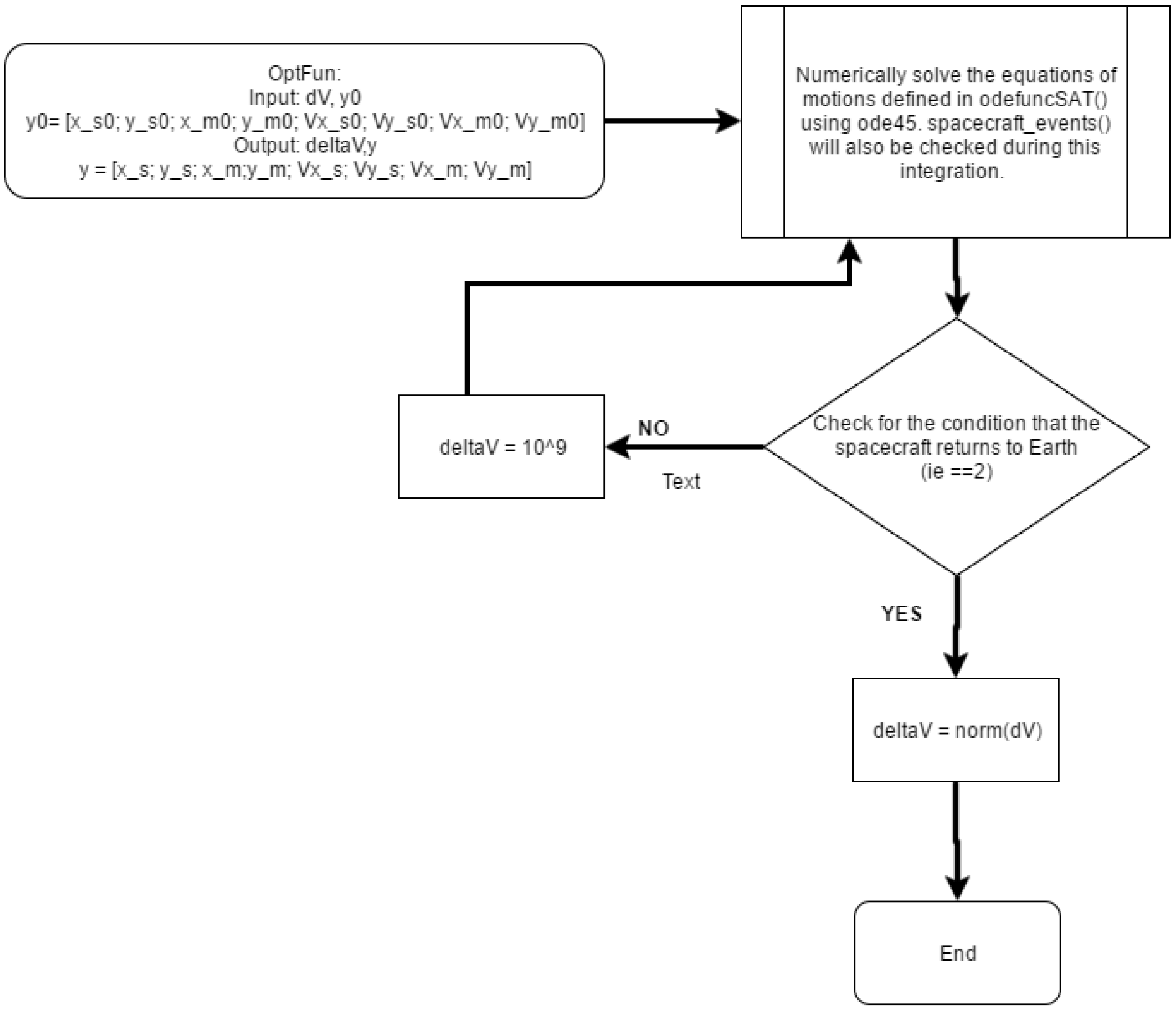
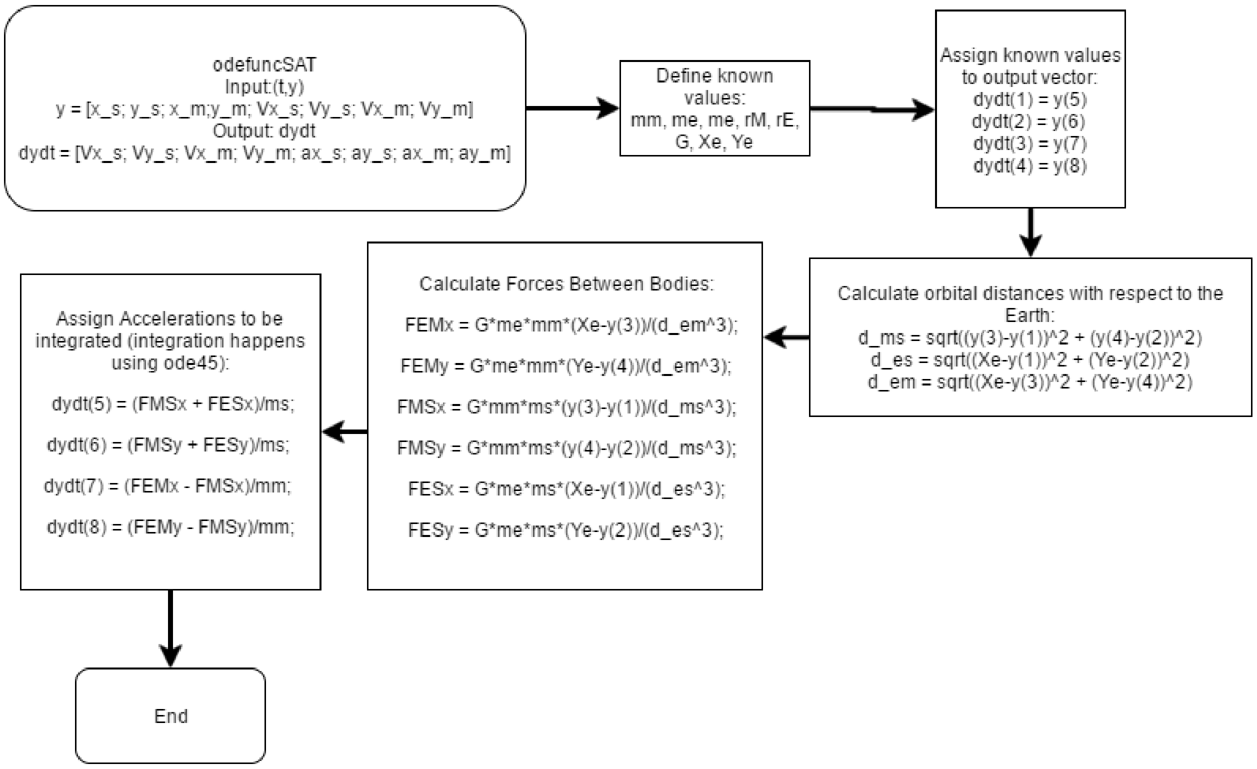

OptFun:
Input: dV, y0
y0= [x_s0; y_s0; x_m0; y_m0; Vx_s0; Vy_s0; Vx_m0; Vy_m0]
Output: deltaV,y
y = [x_s; y_s; x_m;y_m; Vx_s; Vy_s; Vx_m; Vy_m]
Numerically solve the equations of motions defined in odefuncSAT() using ode45. spacecraft_events() will also be checked during this integration.
Check for the condition that the spacecraft returns to Earth (ie ==2)
NO
Text
deltaV = 10^9
YES
deltaV = norm(dV)
End

Figure 2. Function OptFun Flowchart

OptFun takes in the initial guess for deltaV as well as the initial conditions described above. Our specific initial guess was a delta V of $50m/s$. OptFun then uses the matlab solver, ode45, to integrate the equations of motion for the Moon and the spacecraft in an Earth Fixed inertial frame. These equations of motion are defined in the function, odefuncSAT. Another feature to this integration is the the options called spacecraft_events. This function checks for all possible events of the spacecraft, and terminates the numerical integration once one of the conditions is met. After the numerical integration terminates, OptFun checks if the spacecraft returned to Earth. If that condition was met, the program ends, otherwise, the initial guess was changed, and the numerical integration continues. The flowchart for the function, odefuncSAT is shown in Figure (3).


```
odefuncSAT
Input:(t,y)
y = [x_s; y_s; x_m;y_m; Vx_s; Vy_s; Vx_m; Vy_m]
Output: dydt
dydt = [Vx_s; Vy_s; Vx_m; Vy_m; ax_s; ay_s; ax_m; ay_m]

Define known values:
mm, me, me, rM, rE, G, Xe, Ye

Assign known values to output vector:
dydt(1) = y(5)
dydt(2) = y(6)
dydt(3) = y(7)
dydt(4) = y(8)

Calculate orbital distances with respect to the Earth:
d_ms = sqrt((y(3)-y(1))^2 + (y(4)-y(2))^2)
d_es = sqrt((Xe-y(1))^2 + (Ye-y(2))^2)
d_em = sqrt((Xe-y(3))^2 + (Ye-y(4))^2)

Calculate Forces Between Bodies:
FEMx = G*me*mm*(Xe-y(3))/(d_em^3);
FEMy = G*me*mm*(Ye-y(4))/(d_em^3);
FMSx = G*mm*ms*(y(3)-y(1))/(d_ms^3);
FMSy = G*mm*ms*(y(4)-y(2))/(d_ms^3);
FESx = G*me*ms*(Xe-y(1))/(d_es^3);
FESy = G*me*ms*(Ye-y(2))/(d_es^3);

Assign Accelerations to be integrated (integration happens using ode45):
dydt(5) = (FMSx + FESx)/ms;
dydt(6) = (FMSy + FESy)/ms;
dydt(7) = (FEMx - FMSx)/mm;
dydt(8) = (FEMy - FMSy)/mm;

End
```

Figure 3. Function odefuncSAT Flowchart

odefuncSAT takes the initial conditions calculated above, as an input, and redefines some of the known values. Since the required output of the function is the velocity and acceleration vectors of both the Moon and the spacecraft, but the velocities were already an input, they can be renamed, directly into the output vector. The acceleration vector, however, requires further computation. In order to calculate the accelerations, the force components between the spacecraft and the Moon, the Moon and the Earth, and the Earth and the spacecraft must be calculated, as shown in Figure (3). Once the forces are known, Newtons law, shown in Equation (10) can then be applied, and the accelerations can be calculated as shown in Figure (3).

$$F = ma \tag{10}$$

when evaluated by ode45, odefuncSAT terminates when one of the events described in the function spacecraft_events occurs. Figure (4) shows the flowchart for spacecraft_events.

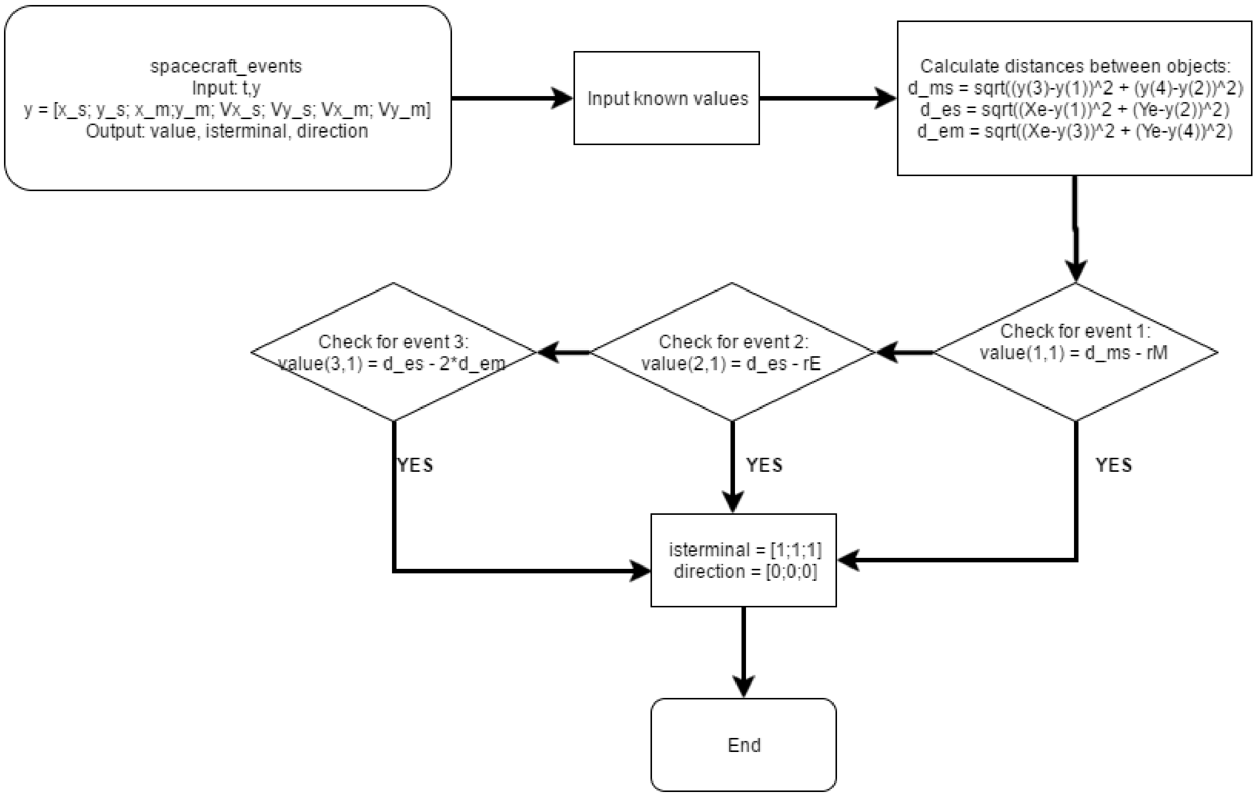

spacecraft_events
Input: t,y
y = [x_s; y_s; x_m;y_m; Vx_s; Vy_s; Vx_m; Vy_m]
Output: value, isterminal, direction
Input known values
Calculate distances between objects:
d_ms = sqrt((y(3)-y(1))^2 + (y(4)-y(2))^2)
d_es = sqrt((Xe-y(1))^2 + (Ye-y(2))^2)
d_em = sqrt((Xe-y(3))^2 + (Ye-y(4))^2)
Check for event 1:
value(1,1) = d_ms - rM
Check for event 2:
value(2,1) = d_es - rE
Check for event 3:
value(3,1) = d_es - 2*d_em
YES
YES
YES
isterminal = [1;1;1]
direction = [0;0;0]
End

Figure 4. Function spacecraft_events Flowchart

There are three possible events that the spacecraft could encounter. The first event is when the spacecraft crashes into the moon. This event occurs when the distance between the center of the moon and the spacecraft is less than or equal to the radius of the Moon. The second event is the desired event. This event occurs when the spacecraft safely returns to Earth. In spacecraft_events, this is defined when the distance between the spacecraft and the Earth is less than or equal to the radius of the Earth. Finally, the third event is the case where the spacecraft is lost to space. This project assumes that this condition is met when the distance between the Earth and the spacecraft is greater than or equal to two times the distance between the Earth and the Moon. If any of these conditions are met, the program is terminated.

B. Simulation Modifications

As seen by the software design setup, the simulation that describes the flight path of the spacecraft is complex. Luckily, the simulation is capable of running many iterations to the problem, and can compare the resulting solutions. This is helpful when done correctly, but can cause issues if the simulation is not set up correctly. For instance, first simulations of this lab reported a ΔV_s of 0 m/s for the spacecraft. Matlab ran the simulation, and found that the least amount of initial velocity required was in fact 0 m/s. However, this is the same simulation case by which the spacecraft collides with the moon, and is obviously not the correct solution. As can be seen in Figure 5, the satellite and moon will collide without a change in initial velocity.

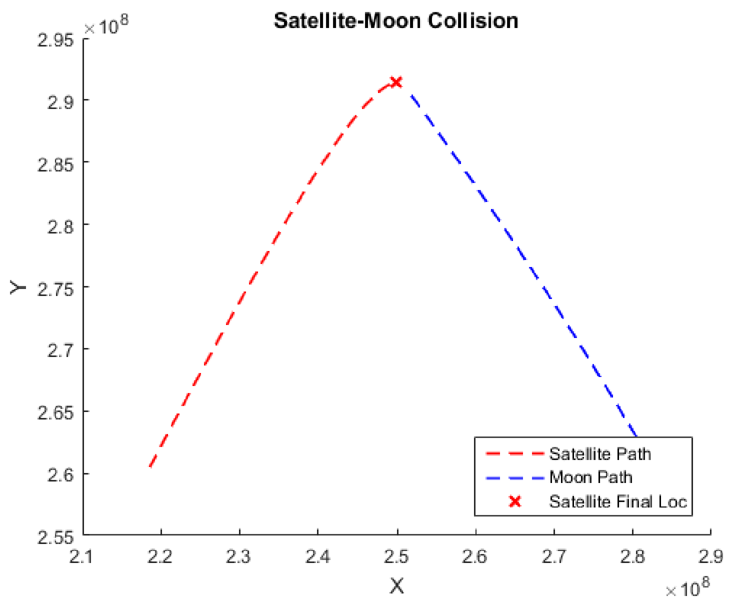


Figure 5. Satellite-Moon collision.

This was the quickest path in time for the spacecraft, but not the ideal solution for this application as crashing a satellite can be quite costly. In response to this, a ”sanity check” was put in place in the space_craft events function to prevent choosing the “crash path” of the satellite. If Matlab converged on this solution again, the ΔV_s for this path was reassigned to a very large number (such as 10^9.) After this change, Matlab no longer chose the “crash course” as the correct solution, because the resulting ΔV_s was so high and no longer a minimum.

C. Simulation Results

From the given algorithm design and the useful simulation modifications, the minimum initial velocity for a safe return to Earth can be calculated. Based on the relative tolerance(s) selected for ode45 calculations, the resulting values for ΔV_s can be found with varying accuracy. Below are tabulated values for the ΔV_s required for a safe return to Earth.

Relative Tolerance	Calculated ΔV_s	Run Time
1e-8	49.1051 m/s	11.57 s
1e-10	49.0980 m/s	18.96 s
1e-13	49.1075 m/s	40.86 s

Increasing the relative tolerance for the ode45 calculations added more runtime to the simulations, but ultimately led to a more accurate solution. This is a common trade-off in numerical analysis, sacrificing performance for speed and vice versa. The physical representation of the minimum ΔV_s required for a safe return home is depicted in Figure 6.

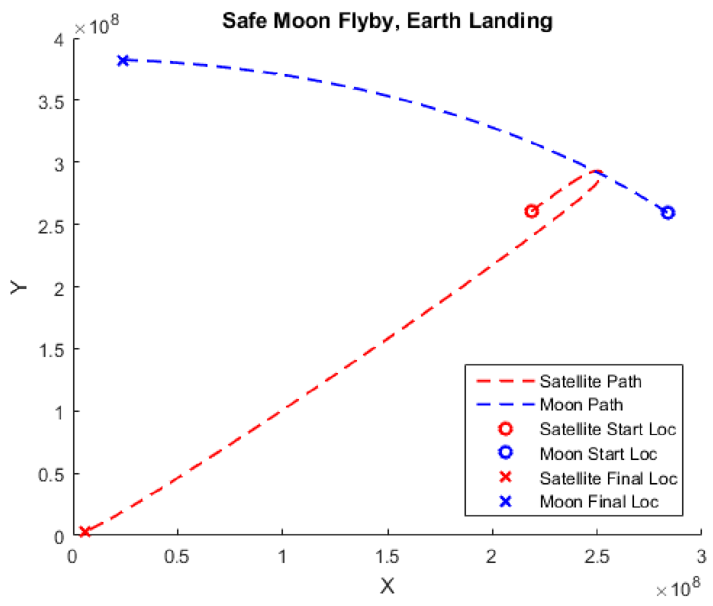


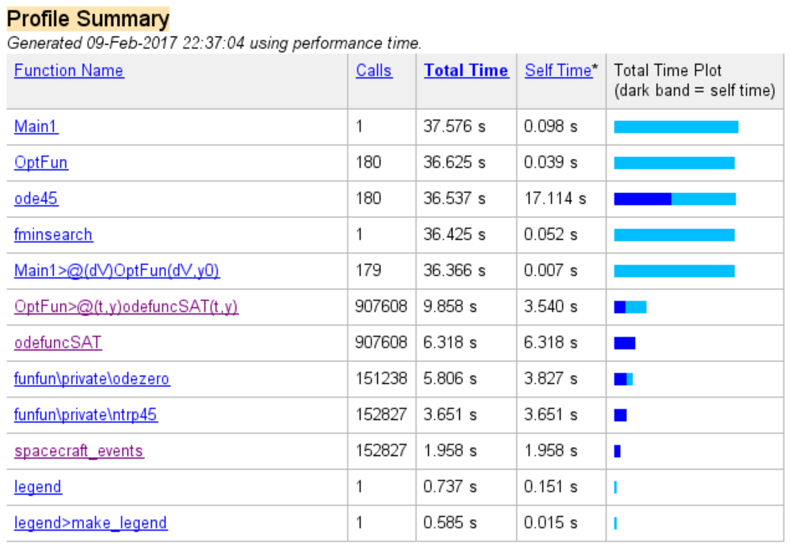
Figure 6. Minimum ΔV_s required solution, resulting in Moon flyby and a safe landing at Earth (origin)

As expected, the simulation depicts the satellite flyby of the moon and Earth landing. The initial velocity required for this flight path is estimated to be 49.1075 m/s.

III. Part 2 - Matlab Software Profiling

A. Profile Summary

The completed simulation performance can be monitored and analyzed using many of Matlab's profiling capabilities. Matlab offers a Profile Summary feature for checking simulation run time, individual function run times, and the number of function calls. The Profile Summary results for this simulation is found in Figure 7.



Profile Summary

Generated 09-Feb-2017 22:37:04 using performance time.

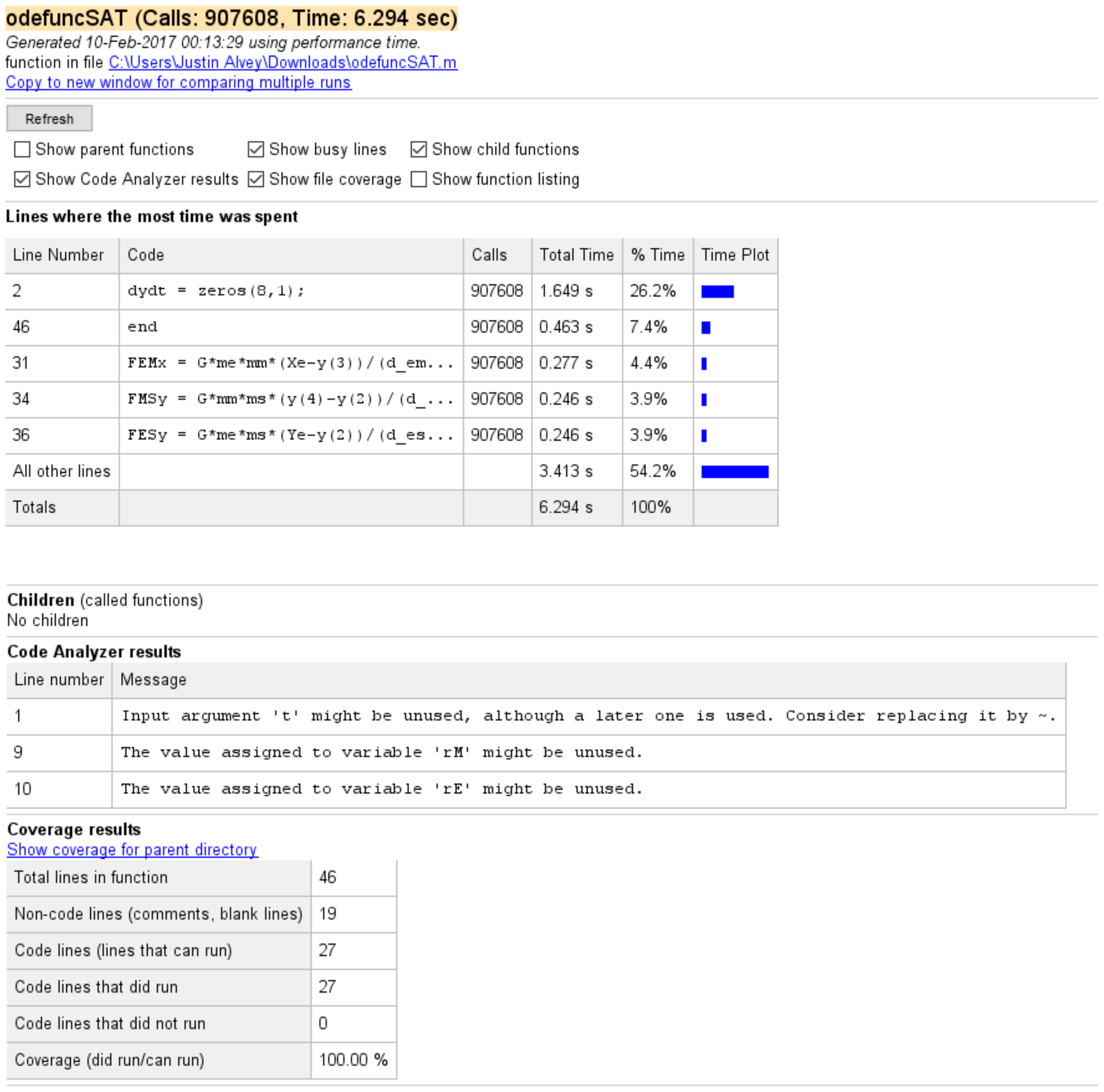
Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Main1	1	37.576 s	0.098 s	
OptFun	180	36.625 s	0.039 s	
ode45	180	36.537 s	17.114 s	
fminsearch	1	36.425 s	0.052 s	
Main1>@(dV)OptFun(dV,y0)	179	36.366 s	0.007 s	
OptFun>@(t,y)odefuncSAT(t,y)	907608	9.858 s	3.540 s	
odefuncSAT	907608	6.318 s	6.318 s	
funfun\private\odezero	151238	5.806 s	3.827 s	
funfun\private\ntrp45	152827	3.651 s	3.651 s	
spacecraft_events	152827	1.958 s	1.958 s	
legend	1	0.737 s	0.151 s	
legend>make_legend	1	0.585 s	0.015 s	

Figure 7. Function spacecraft_events Flowchart

From the Profile Summary results, calling ode45 is the most computationally expensive function call in the program at 16.98 seconds. Calling the ode45 function takes up over 45% of the program's total run time, which is plausible since ode45 is the main solver used in this simulation. The ode45 function is thus very expensive computationally, and adds more run time by calling outside functions for help such as odezero and ntrp45. However, since these functions are built-in to Matlab and foundational to solving simulations, they will not be looked at in detail for the sake of time.

B. Function: odeFuncSat Detail Report

Outlined in Figure 8 are the profiling results for the function odefuncSat.



odefuncSAT (Calls: 907608, Time: 6.294 sec)

Generated 10-Feb-2017 00:13:29 using performance time.
function in file C:\Users\Justin Alvey\Downloads\odefuncSAT.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

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
2	`dydt = zeros(8,1);`	907608	1.649 s	26.2%	▬
46	`end`	907608	0.463 s	7.4%	▪
31	`FEMx = G*me*mm*(Xe-y(3))/(d_em...`	907608	0.277 s	4.4%	▪
34	`FMSy = G*mm*ms*(y(4)-y(2))/(d_...`	907608	0.246 s	3.9%	▪
36	`FESy = G*me*ms*(Ye-y(2))/(d_es...`	907608	0.246 s	3.9%	▪
All other lines			3.413 s	54.2%	▬
Totals			6.294 s	100%	

Children (called functions)
No children

Code Analyzer results

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

Coverage results
Show coverage for parent directory

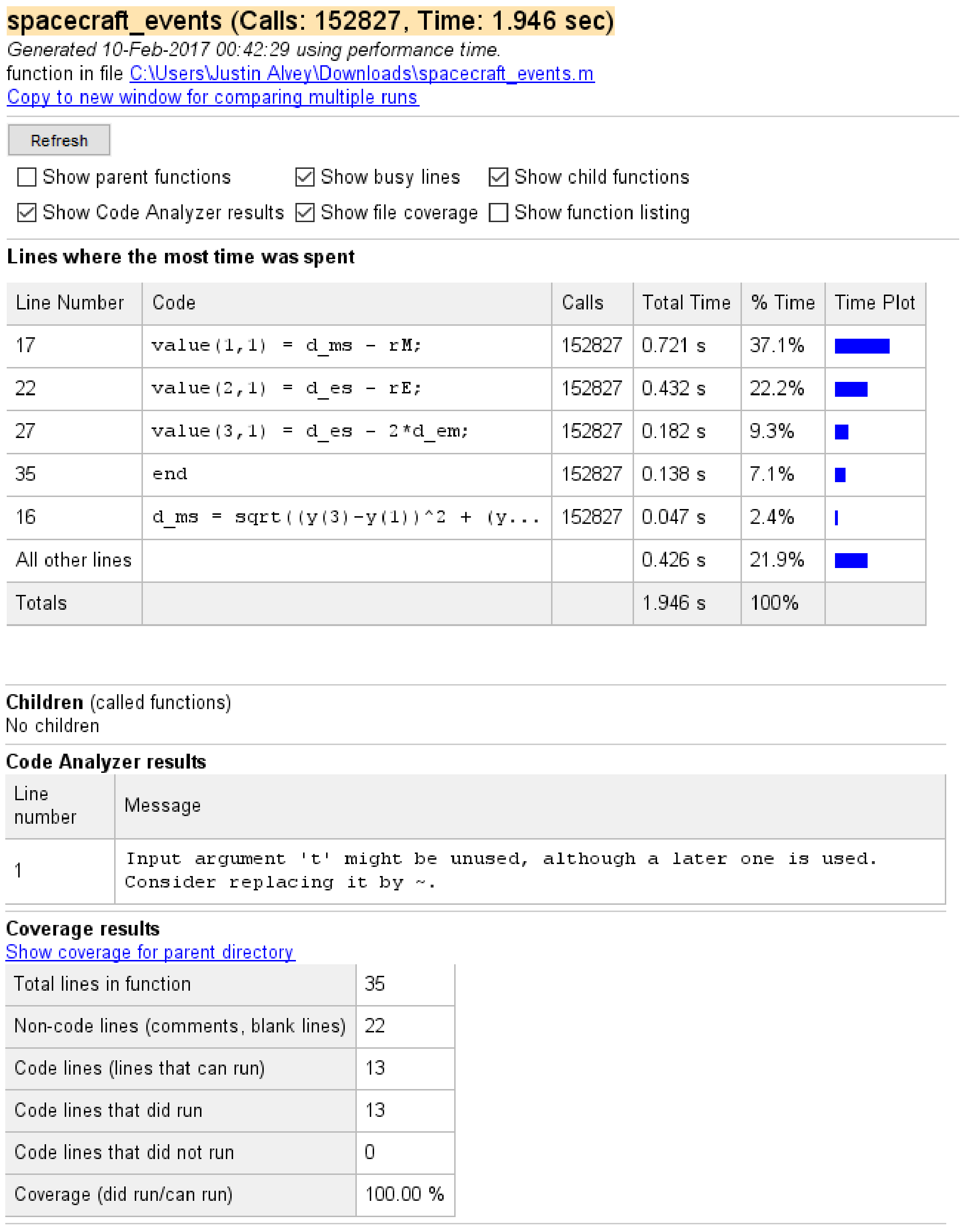
Total lines in function	46
Non-code lines (comments, blank lines)	19
Code lines (lines that can run)	27
Code lines that did run	27
Code lines that did not run	0
Coverage (did run/can run)	100.00 %

Figure 8. Profiling breakdown for odefuncSat.m function in Matlab

OdefuncSat is the most expensive external function in this simulation. Busy lines for this function include initializing the dydt array on line 2, and calculating the forces between celestial bodies in question on lines 31, 34, and 36. This function has no children functions. For improved computing efficiency, the number of variables in the function could be limited by excluding variables that are currently not being used. The code analyzer displays these cases, which appear on lines 1, 9, and 10. Coverage results also show 100% coverage, as 27 code lines that can run are run in the function.

C. Function: spacecraft_events Detail Report

Outlined in Figure 9 are the profiling results for the function spacecraft_events.



spacecraft_events (Calls: 152827, Time: 1.946 sec)

Generated 10-Feb-2017 00:42:29 using performance time.
function in file C:\Users\Justin Alvey\Downloads\spacecraft_events.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

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
17	`value(1,1) = d_ms - rM;`	152827	0.721 s	37.1%	■
22	`value(2,1) = d_es - rE;`	152827	0.432 s	22.2%	■
27	`value(3,1) = d_es - 2*d_em;`	152827	0.182 s	9.3%	■
35	`end`	152827	0.138 s	7.1%	■
16	`d_ms = sqrt((y(3)-y(1))^2 + (y...`	152827	0.047 s	2.4%	▏
All other lines			0.426 s	21.9%	■
Totals			1.946 s	100%	

Children (called functions)
No children

Code Analyzer results

Line number	Message
1	`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	35
Non-code lines (comments, blank lines)	22
Code lines (lines that can run)	13
Code lines that did run	13
Code lines that did not run	0
Coverage (did run/can run)	100.00 %

Figure 9. Profiling breakdown for spacecraft_events.m function in Matlab

Spacecraft_events is another expensive external function for this simulation, which is included in the ode45 call to check event parameters. Busy lines for this function include calculating the value checks for the three termination cases: colliding with the moon, landing on the Earth, and missing the Earth. This function has no children functions. The code analyzer has caught only one issue, where the input argument t is unused. Coverage results also show 100% coverage, as 13/13 code lines that can run are run in the function.

D. Profiling Results and Runtime Modifications

Results of the Matlab profiler and its many profiling capabilities are far-reaching. The Matlab profiler itself is a snapshot of useful code statistics and overall execution information. The most important results of the Matlab Profiler are the run times of specific functions, and the number of function calls. Based on this information, the performance of code can be assessed accurately and shortcuts taken to improve this. Although the run times of some methods are limited by the back-end setup (such as ode45 which is native to Matlab), non-essential tasks can be limited to improve code speed.

To improve run time, other time integration methods can be implemented. Numerical solvers exist, such as ode15s ode23s, that are crucial upgrades compared to ode45 for "stiffer" systems. These functions can speed up solver tasks that ode45 takes longer to complete, as its solver techniques are more general and not "stiff-specific." Regardless, ode45 is an excellent solver for most integration requirements. Ultimately, Matlab is an interpretive language that is easy to use and useful for developing complex simulations and other algorithms. However for speed and durability, other languages and platforms that allow compiling will always be faster in completing calculations. Matlab serves its purpose as useful language for interpretive tasks and undoubtedly be used for a long time in academia and data analysis for its out-of-the box functionality and wide database of built-in functions.