Robert Hakulin
Ankit Hriday
ASEN 4057 Assignment 6

Explanation of the Design Process

Numerical Integrator

The integrating method that was used was the first order Euler's method presented in the Assignment 6 documentation. The Euler's method is a simple and quick integration method but it does lack accuracy do to first order constraint. Smaller time step can give a reasonably accurate number which is what we use. We calculate the forces every iteration which gives us the acceleration and hence the velocity.

Optimization Algorithm

The optimization function that was used was a 2 step grid search varying the x and y velocities. The first step was a rough, coarse resolution grid search that found the minimum value within 5 m/s. After, a second grid search is performed using the tolerance that is input from the user in order to narrow down the minimum value to within the tolerance. This helps us speed up the code as doing a grid search for everything within the tolerance will slow down the code considerably.

Plotting Algorithm

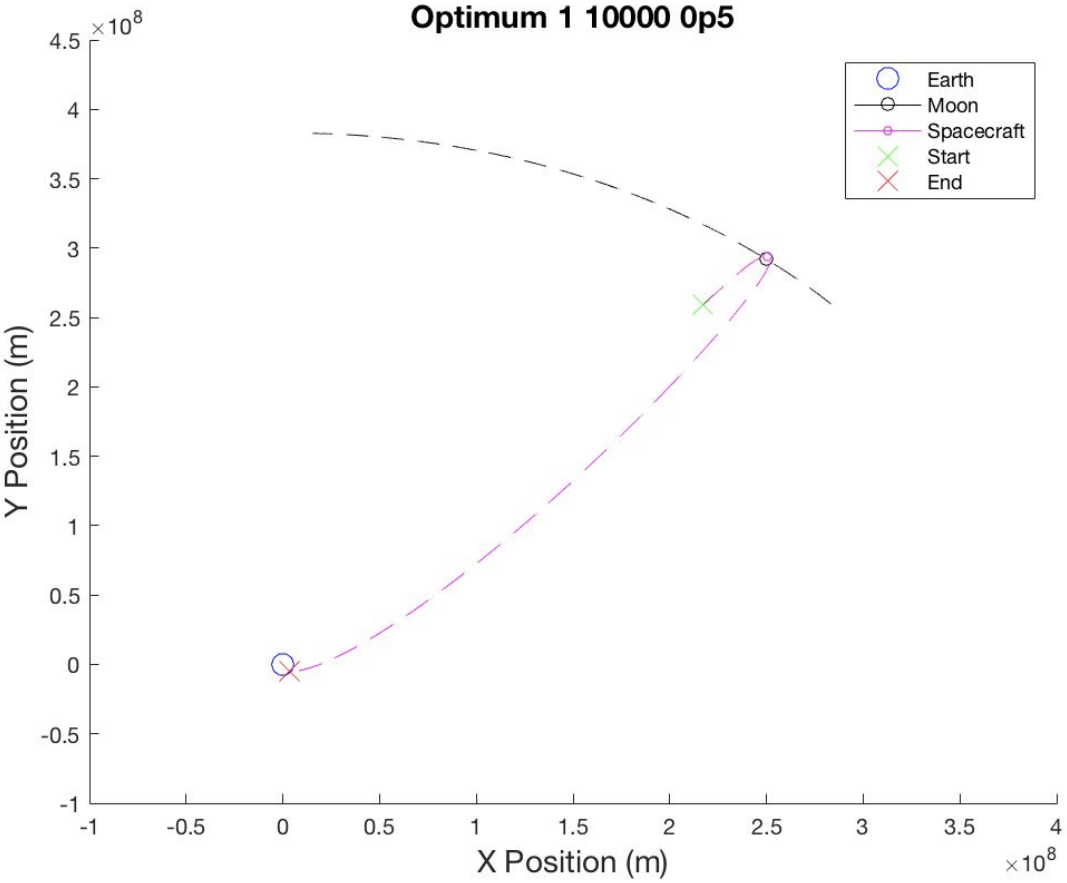
To plot the final trajectories, an output text file is made using the positions of the 3 bodies. The file is then input to a matlab script that will plot the desired trajectory. The output text file is named in order of which objective is getting solved for and what is the desired time step. We then save all these values in a matlab friendly format. Plot is generated from MATLAB.

Source Files

We didn't use a source file from outside. We only used the dedicated math library which we call using -lm in our script and also the stdio.h

Part-2 Plotting

Objective-1

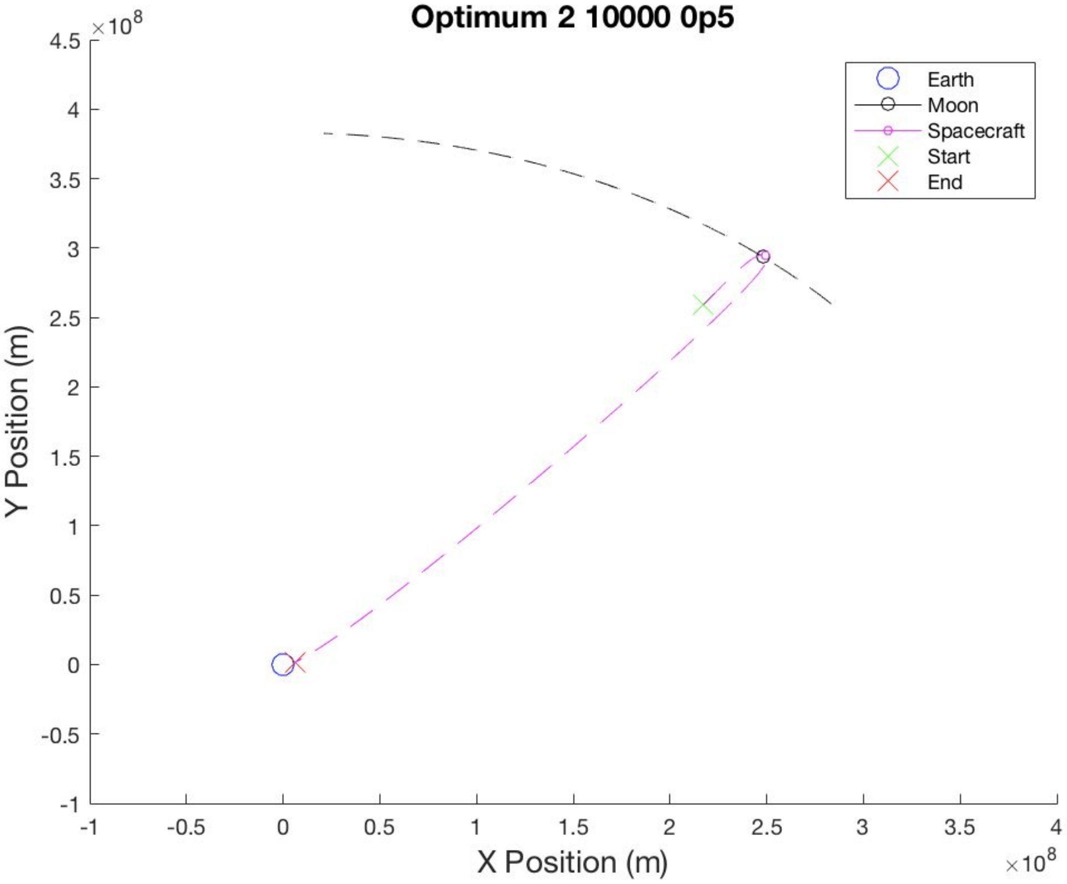


Output

```
Solving for Objective 1...

Grid searching all values with course resolution...

Initial Guess for objective 1:    DVX = 0.000 m/s
                                  DVY = 80.000 m/s

Searching with fine resolution near initial guess...

Final values for objective 1:    DVX = 0.000 m/s
                                 DVY = 78.500 m/s

The minimum delta V to return to Earth is 78.500 m/s
The minimum delta V = 78.500 m/s
```

```
Integration Result:    The Spacecraft has returned to Earth

Final Iteration Number is:        9987
Time is:                          3.459 days
Final Spacecraft-Earth Distance:    6255024 m
Final Spacecraft-Moon Distance:        388023369 m
Final Moon-Earth Distance:     383027563 m
```

Objective-2



Output

```
Solving for Objective 2...

Grid searching all values with course resolution...

Initial guess:    DVX = -62.000 m/s
                  DVY = 78.000 m/s

Searching with fine resolution near initial guess...
```

Final values for objective 2: DVX = -62.500 m/s
DVY = 78.000 m/s

The minimum time to return to Earth is 3.480 days
The minimum delta V = 99.951 m/s

Integration Result: The Spacecraft has returned to Earth

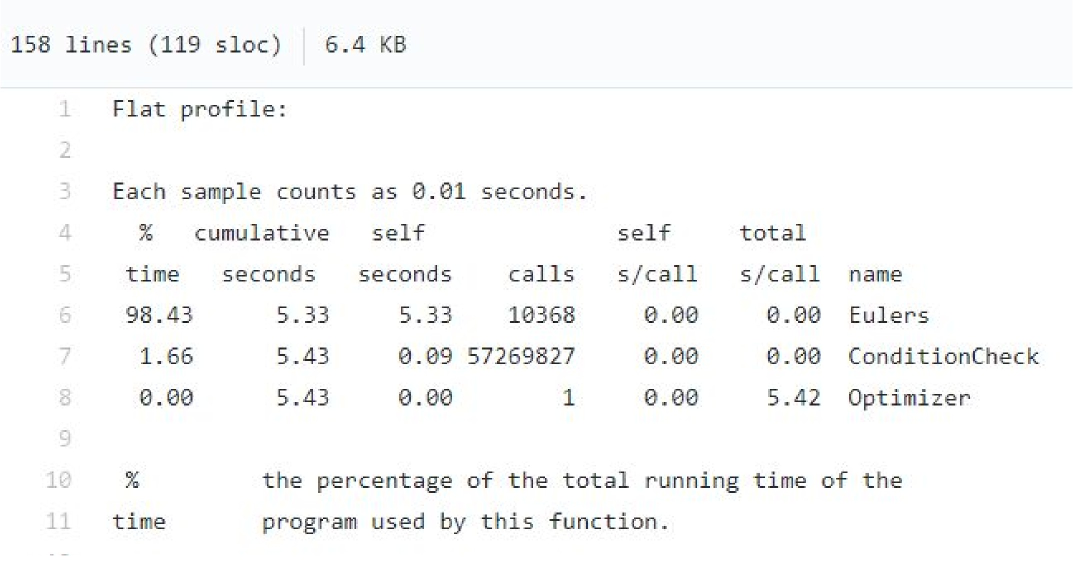
Final Iteration Number is: 9802
Time is: 3.394 days
Final Spacecraft-Earth Distance: 6256360 m
Final Spacecraft-Moon Distance: 381183597 m
Final Moon-Earth Distance: 383075805 m

Part-3 Bash script

The bash script can be found under the filename: **Assignment6.sh**. This when run generates the output which is saved in the Output folder.

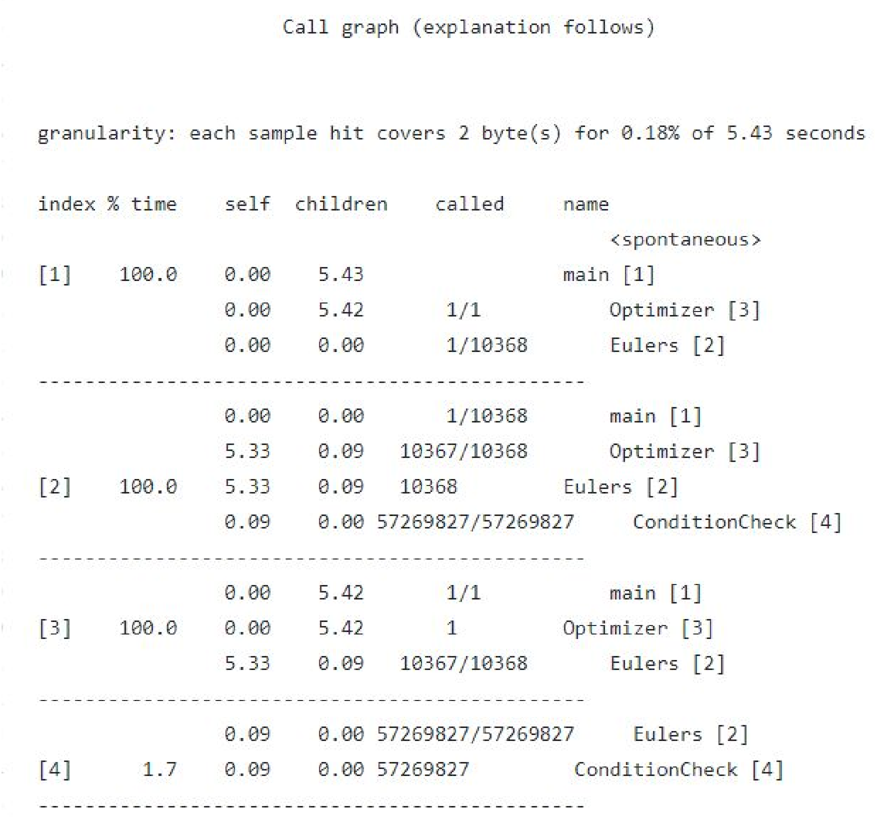
Profile Report

Objective-1



```
158 lines (119 sloc) | 6.4 KB

Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 98.43      5.33     5.33    10368     0.00     0.00  Eulers
  1.66      5.43     0.09 57269827     0.00     0.00  ConditionCheck
  0.00      5.43     0.00        1     0.00     5.42  Optimizer

 %         the percentage of the total running time of the
time       program used by this function.
```

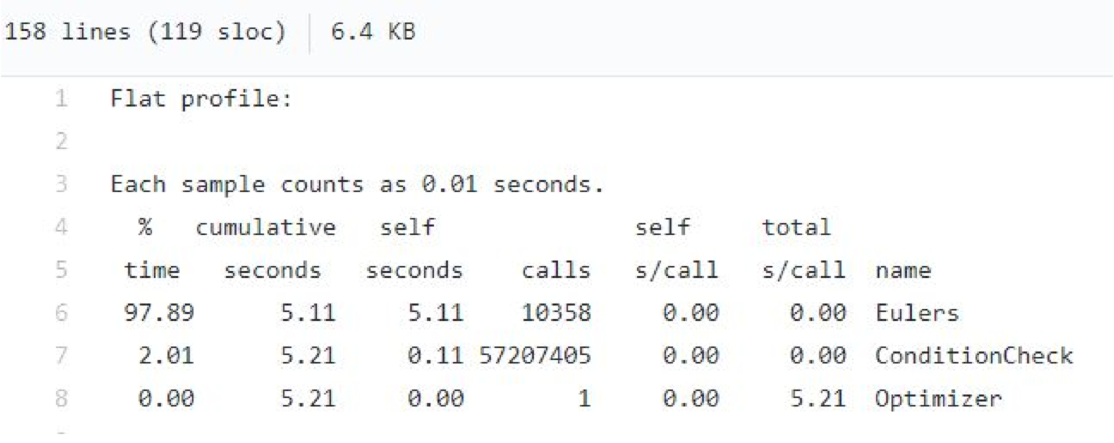
Fig: Objective-1 Flat Profile



```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.18% of 5.43 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    5.43                 main [1]
                0.00    5.42       1/1           Optimizer [3]
                0.00    0.00       1/10368       Eulers [2]
-----------------------------------------------
                0.00    0.00       1/10368       main [1]
                5.33    0.09   10367/10368       Optimizer [3]
[2]    100.0    5.33    0.09   10368         Eulers [2]
                0.09    0.00 57269827/57269827     ConditionCheck [4]
-----------------------------------------------
                0.00    5.42       1/1           main [1]
[3]    100.0    0.00    5.42       1         Optimizer [3]
                5.33    0.09   10367/10368       Eulers [2]
-----------------------------------------------
                0.09    0.00 57269827/57269827     Eulers [2]
[4]      1.7    0.09    0.00 57269827         ConditionCheck [4]
-----------------------------------------------
```

Fig: Objective-1 Call Graph

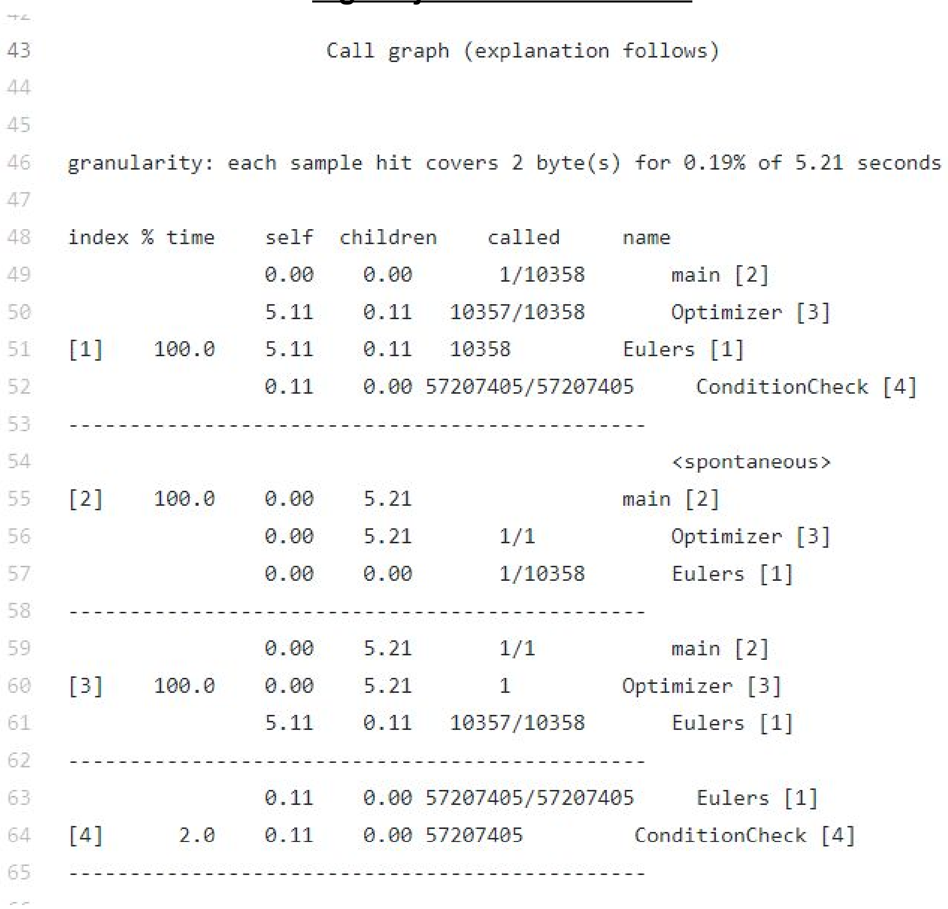
Objective-2



158 lines (119 sloc) | 6.4 KB

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 97.89      5.11     5.11    10358     0.00     0.00  Eulers
  2.01      5.21     0.11 57207405     0.00     0.00  ConditionCheck
  0.00      5.21     0.00        1     0.00     5.21  Optimizer
```

Fig: Objective-2 Flat Profile



```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.19% of 5.21 seconds

index % time    self  children    called     name
                0.00    0.00       1/10358       main [2]
                5.11    0.11   10357/10358       Optimizer [3]
[1]    100.0    5.11    0.11   10358         Eulers [1]
                0.11    0.00 57207405/57207405     ConditionCheck [4]
-----------------------------------------------
                                                 <spontaneous>
[2]    100.0    0.00    5.21                 main [2]
                0.00    5.21       1/1           Optimizer [3]
                0.00    0.00       1/10358       Eulers [1]
-----------------------------------------------
                0.00    5.21       1/1           main [2]
[3]    100.0    0.00    5.21       1         Optimizer [3]
                5.11    0.11   10357/10358       Eulers [1]
-----------------------------------------------
                0.11    0.00 57207405/57207405     Eulers [1]
[4]      2.0    0.11    0.00 57207405         ConditionCheck [4]
-----------------------------------------------
```

Fig: Objective-2 Call Graph

For both objectives from the flat profile we can see that Eulers take the most time. This is because everytime we perform a grid search we have to Euler , this in turn also calls the ConditionCheck looked from the Call Graph. So, the times they are called are the same. Euler's take the most time to compute as in Euler's we derive all the euler equatiopns for a given velocity while in ConditionCheck is only check of conditions and doesn't do a lot of caluclations. Hence, Eulers takes about 98% of the time the code runs for.