Table of Contents


```
%-------- HW 7 MATLAB code --------%
% Romeo Perlstein, section 0101    %
% Chat it's over... I'm cooked!!!  %

% Hey Eric (I think), or I mean Dr. Frizzell! Hope things have been good,
% PLEASE wish me luck in this class as I am STRUGGLING (Idk why it's been
% so hard to do school work I think I'm just cringe. Also I think Prof.
% Barbee doesn't like me heheh. OH WELL I guess the semester is over

close all
```

Q1

```
mew_earth = 0.39860*10^6; % km^3/s^2

% Equitorial, spherical orbit - velocity is constant
spacecraft_position_initial_1 = [8000;0;0]; % km
```

a

```
spacecraft_velocity_initial_1 = [0;sqrt(mew_earth/
spacecraft_position_initial_1(1));0];
accel_thrust1 = 1*10^-4; % kN/kg
```

b

see attached PDF

c

```
t01 = 0;
term1_1 = (norm(spacecraft_velocity_initial_1)/accel_thrust1);
term2_1 = (20*((accel_thrust1)^2)*(norm(spacecraft_position_initial_1)^2))/
(norm(spacecraft_velocity_initial_1)^4);
tesc1 = term1_1*( 1-term2_1^(1/8) );
```

```
fprintf("Time to escape, in seconds (analytical):\n");
tesc1
fprintf("Time to escape, in minutes (analytical):\n");
tesc1/60
```

d

make a da graph-a

```
%--- ODE func values from HW1---%
tall_er_ant = (10^-13); % Tolerance
step_size = 1; % step size
max_time = 2*tesc1; % max time (0->max_time)
t = [0:step_size:max_time]; % timestep

% ODE options
ODE_options = odeset("RelTol", tall_er_ant, "AbsTol", tall_er_ant);

initial_state = [spacecraft_position_initial_1;spacecraft_velocity_initial_1];


[T1, Y1] = ode45(@myodefun, t, initial_state, ODE_options, mew_earth,
 accel_thrust1);

hold on
plot(0,0, ".b", "MarkerSize", 50, "DisplayName","Earth")
plot(Y1(:,1), Y1(:,2), "-r","DisplayName","s/c orbit")
plot(spacecraft_position_initial_1(1),
 spacecraft_position_initial_1(2), ".k", "MarkerSize",
 10, "DisplayName", "spacecraft")
title("Graph of integrated orbit from the given initial conditions (Q1)")
xlabel("X position");
ylabel("Y position")
legend
grid on
axis equal
fprintf("The final velocity of the propegated orbit is:\n")
len1 = length(Y1(:,4:6));
Y1(len1,4:6)
```

e

find r_esc, then find time using prop'd data

```
term1_1_1 =
 norm(spacecraft_position_initial_1)*norm(spacecraft_velocity_initial_1);
term2_1_1 = 20*(accel_thrust1^2)*(norm(spacecraft_position_initial_1)^2);
resc1 = term1_1_1/(term2_1_1^(1/4));
fprintf("Radius of escape:\n")
resc1

tall_er_ant2 = 1;
for i=1:1:max_time+1
```

```
    spef_energy(i) = (norm(Y1(i,4:6))^2)/2 - mew_earth/norm(Y1(i,1:3));
    if(spef_energy(i) > 0-0.0001 && spef_energy(i) < 0+0.0001)
        time_of_energy_switch = i;
    end
    if ((norm(Y1(i,1:3)) < resc1+1) && (norm(Y1(i,1:3)) > resc1-1))
        time_at_esc_radius = i;
    end
end
figure
hold on
plot(T1, spef_energy)
yline(0, "-k")
grid on
title("Plot of specific energy vs Time")
xlabel("Time (seconds)")
ylabel("Specific Energy")


fprintf("Time to reach calculated escape radius (seconds):\n")
time_at_esc_radius
fprintf("Time of escape calculated (seconds):\n")
tesc1
fprintf("Difference in times (seconds):\n")
time_at_esc_radius - tesc1
fprintf("Time when energy flips from negative to positive (time when it is 0)
 (seconds):\n")
time_of_energy_switch


% From inspection of the integrated data, r_esc is reached at 36413
% seconds, different from our analytical answer of 34047 seconds, differing
% by roughly 2366 seconds (almost an hour!). However, inspection of the
% change in specific energy's sign value (i.e., when specific energy's
% value goes from negative to positive, or when it cross the x-axis) occurs
% at roughly 50130 seconds, which is about 20,000 seconds more than our
% calculated escape time!

% Both numerically found escape times are greater than the analytical
% answer. I think the analytical answer is conservative with it's output
% because it really doesn't take into account the force of gravity on the
% spacecraft the same way that the numerical integrated does. Since it's
% only going off of the kinematics of the problem (other than the
% acceleration due to thrust being a non-kinematic term since it is a
% force), it does not account for the extra time it might take for the
% spacecraft to break from the planets gravity well, giving us a
% conservative value.
%
% TL:DR - The exclusion of the gravity term from the t_esc equation leads
% to a faster escape time being found. Since the t_esc equation is only
% influenced by a single force (and not the accounting for the
% gravitiational force being applied to the craft), it will find a faster
% escape time.
```

Do everything like 10 times.... hurray... FOR LOOP TIME BAYBE

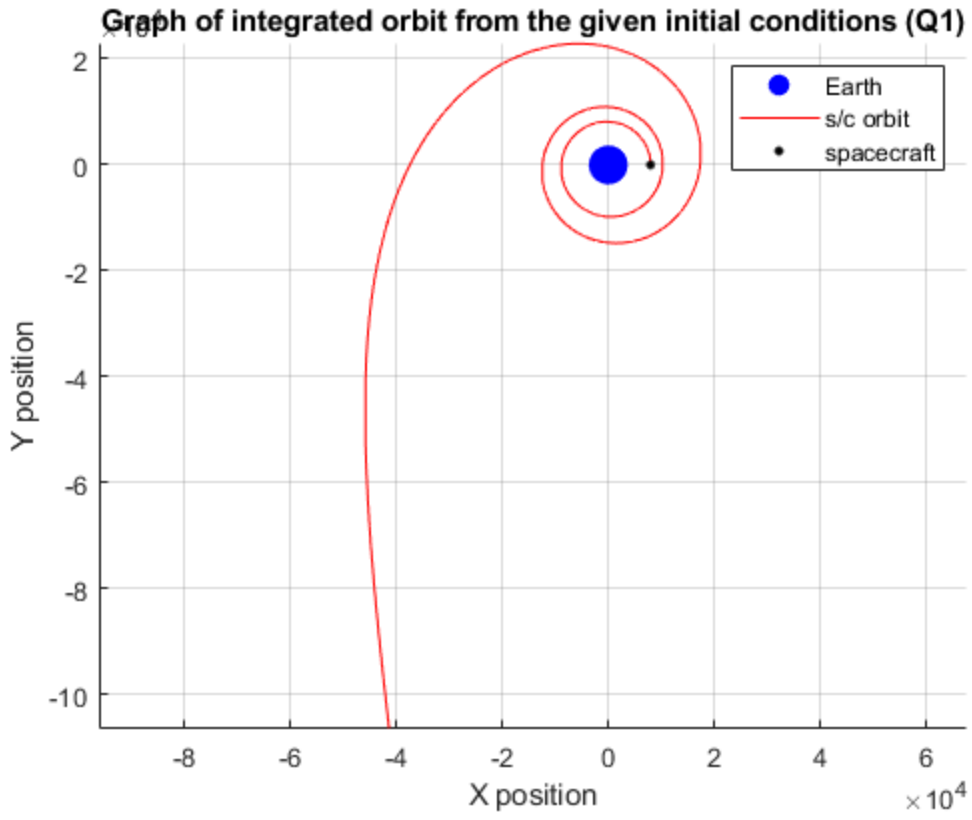
```
accel_thrusts1 =
 [0.00015;0.00025;0.00035;0.00045;0.00055;0.00065;0.00075;0.00085;0.00095;0.001];
loading_time = "Loading: [";
for i=1:1:10
    accel_thrust = accel_thrusts1(i);
    term1_1 = (norm(spacecraft_velocity_initial_1)/accel_thrust);
    term2_1 = (20*((accel_thrust)^2)*(norm(spacecraft_position_initial_1)^2))/
(norm(spacecraft_velocity_initial_1)^4);
    tesc1_multi(i) = term1_1*( 1-term2_1^(1/8) );

    max_time = 10*tesc1; % max time (0->max_time)
    t = [0:step_size:max_time]; % timestep

    [T1, Y1] = ode45(@myodefun, t, initial_state, ODE_options, mew_earth,
 accel_thrust);

    term1_1_1 =
 norm(spacecraft_position_initial_1)*norm(spacecraft_velocity_initial_1);
    term2_1_1 = 20*(accel_thrust^2)*(norm(spacecraft_position_initial_1)^2);
    resc1 = term1_1_1/(term2_1_1^(1/4));

    for ii=1:1:max_time+1
        spef_energy_multi(ii) = (norm(Y1(ii,4:6))^2)/2 - mew_earth/
norm(Y1(ii,1:3));
        if(spef_energy_multi(ii) > 0-0.001 && spef_energy_multi(ii) < 0+0.001)
            time_of_energy_switch_multi(i) = ii;
        end
        if ((norm(Y1(ii,1:3)) < resc1+1) && (norm(Y1(ii,1:3)) > resc1-1))
            time_at_esc_radius_multi(i) = ii;
        end
    end
    loading_time = loading_time + "=";
    fprintf(loading_time + "]\n")
end

if(length(time_at_esc_radius_multi) ~= 10)
    fprintf("You effed up!")
end
if(length(time_of_energy_switch_multi) ~= 10)
    fprintf("You effed up boy!")
end
figure
hold on
title("Plot of escape times using Log scale")
plot([1:1:10], tesc1_multi, "-r", DisplayName="Analytical t_e_s_c")
plot([1:1:10], time_at_esc_radius_multi, "-b", DisplayName="Numerical time
 Value for solved r_e_s_c")
plot([1:1:10], time_of_energy_switch_multi, "-m", DisplayName="Numerical Value
 for time at energy flip")
xlabel("interation")
ylabel("Time (seconds)")
grid on
set(gca,"yscale","log")
legend
```

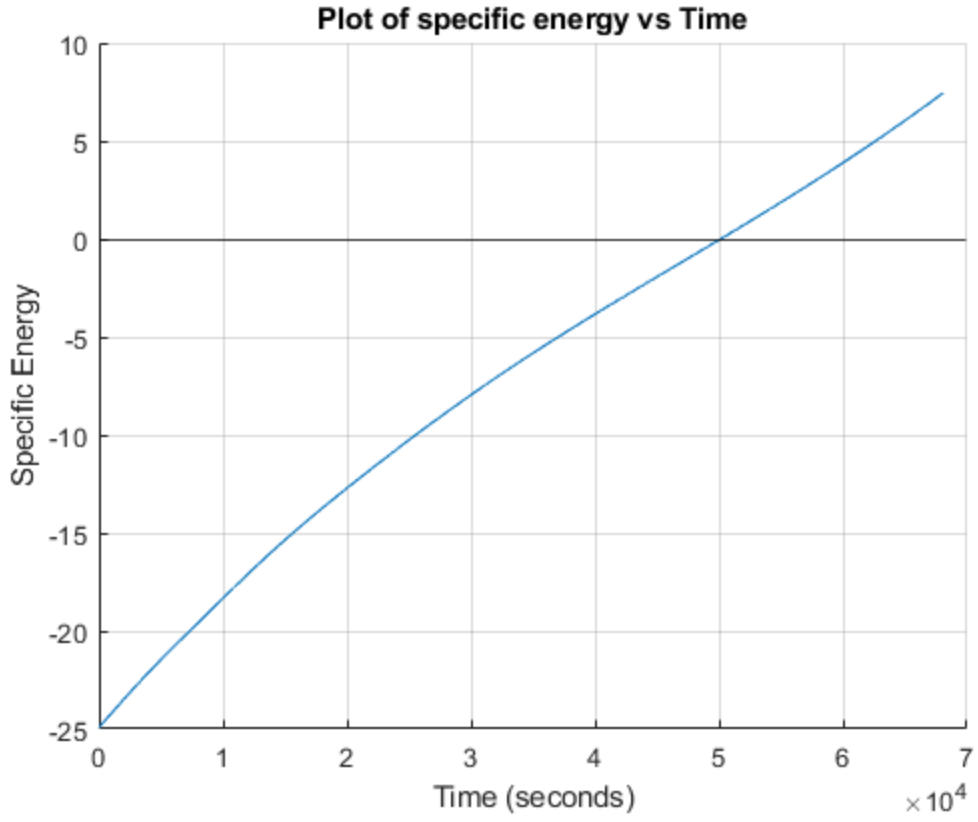
g

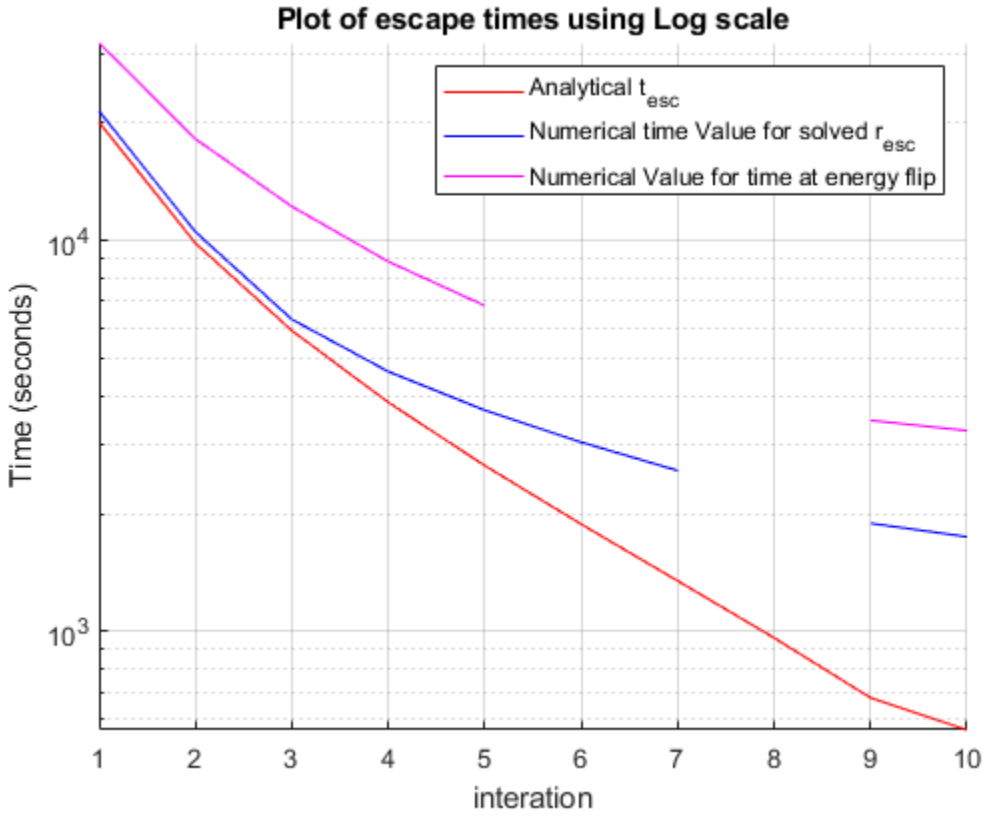
From the plot, it is clear that the analytical method is lacking in it's accuracy. As the acceleration due to thrust is increased, we see a larger separation of numerical vs analytical values of the escape time. However, there are two key things to note: the analytical method produces a smooth, continuous curve of solutions, because it gives you exactly one solution. The integration method requires that you find the value at which either the energy changes or the time at which you reach your analytically solved for r_esc. Disregarding the outliers of data in the provided plot, and assuming they are smooth curves, we can still see the growing gap between the analytical solution and the numerical integration soltuion. The main limitation, I believe of the analytical method is that it does not incorporate gravity into the equation at all, and only relies on the input thrust as it's only force. I think this limitation is what causes the analytical method to undershoot, as it requires less time to escape if there is no modeled force of gravity "hold you back" (resisting the spacecrafts power to escape orbit)

```
Time to escape, in seconds (analytical):

tesc1 =

     3.404728618711480e+04

Time to escape, in minutes (analytical):

ans =

     5.674547697852466e+02

The final velocity of the propegated orbit is:

ans =

   0.539602051389340  -4.648690556058700                   0

Radius of escape:

resc1 =

     2.985459254170815e+04

Time to reach calculated escape radius (seconds):

time_at_esc_radius =

       36413

Time of escape calculated (seconds):

tesc1 =

     3.404728618711480e+04

Difference in times (seconds):

ans =
```

```
    2.365713812885202e+03

Time when energy flips from negative to positive (time when it is 0)
 (seconds):

time_of_energy_switch =

       50130

Loading: [=]
Loading: [==]
Loading: [===]
Loading: [====]
Loading: [=====]
Loading: [======]
Loading: [=======]
Loading: [========]
Loading: [=========]
Loading: [==========]
```




Plot of specific energy vs Time
Specific Energy
Time (seconds)
×10^4
10
5
0
-5
-10
-15
-20
-25
0
1
2
3
4
5
6
7


Plot of escape times using Log scale
Analytical t_esc
Numerical time Value for solved r_esc
Numerical Value for time at energy flip
Time (seconds)
interation
10^4
10^3
1
2
3
4
5
6
7
8
9
10

Q3

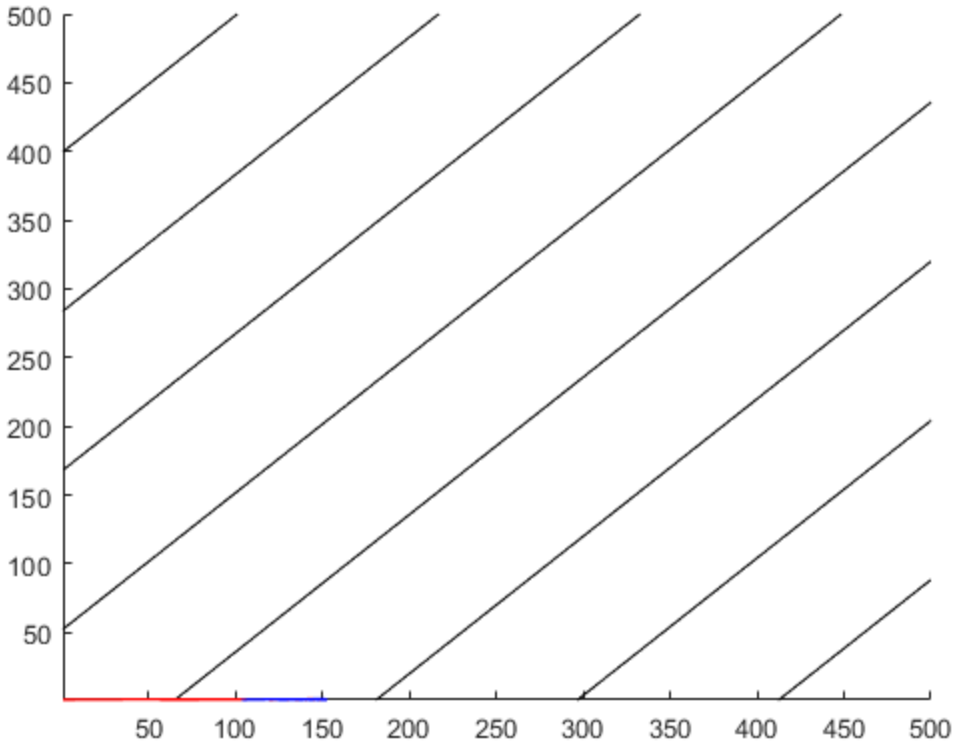
```
mew_sun = 132712*10^6; % km^3/s^2

% Create a 2D matrix containing all of our TOF values
starting_depart_date = ymdhms2jd(2022, 8, 1, 12, 0, 0);
starting_arrival_date = ymdhms2jd(2023, 1, 28, 12, 0, 0);

TOF_matrix(500,500) = 0;
Julian_matrix(500,500) = 0;
for i=0:1:499
    for ii=0:1:499
        TOF_matrix(i+1,ii+1) = ((starting_arrival_date+i) -
 (starting_depart_date+ii))*24*60*60;
    end
end

V_infinity_matrix(500,500) = 0;
C3_martix(500,500) = 0;
% for i=0:1:499
%     for ii=0:1:499
%         TOF1 = TOF_matrix(i+1,ii+1);
%         [r1_vec1, vel_earth] = findEarth(starting_depart_date+ii);
%         [r2_vec1, vel_mars] = findMars(starting_arrival_date+i);
%         r1_1 = norm(r1_vec1);
%         r2_1 = norm(r2_vec1);
%         cos_deltaV1 = (dot(r1_vec1,r2_vec1))/(r1_1*r2_1);
%         DM1 = 1; % Given that the thang is short way
%         A1 = DM1*sqrt(r1_1*r2_1*(1+cos_deltaV1));
%
%         % Weird starter numbers I guess?
%         trident1 = 0;
%         C2_1 = 1/2;
%         C3_1 = 1/6;
%         trident_hp1 = 4*(pi^2);
%         trident_low1 = -4*(pi^2);
%         deltat1 = 0;
%         tolerance = 10^-5; % good tolerance
%         while(abs(TOF1-deltat1) >= tolerance)
%             y1 = r1_1 + r2_1 + (A1*((trident1*C3_1)-1)/sqrt(C2_1));
%             if(A1>0 && y1 <0)
%                 trident_low1 = trident_low1 + (pi/4);
%             end
%             x1 = sqrt(y1/C2_1);
%             deltat1 = (((x1^3)*C3_1)+(A1*sqrt(y1)))/sqrt(mew_sun);
%             if deltat1 < TOF1
%                 trident_low1 = trident1;
%             else
%                 trident_hp1 = trident1;
%             end
%             trident1 = (trident_hp1 + trident_low1)/2;
%             if trident1 > tolerance
%                 C2_1 = (1-cos(sqrt(trident1)))/trident1;
```

```
%                 C3_1 = (sqrt(trident1)-sin(sqrt(trident1)))/
sqrt(trident1^3);
%             elseif trident1 < -tolerance
%                 C2_1 = (1-cosh(sqrt(-trident1)))/trident1;
%                 C3_1 = (sinh(sqrt(-trident1))-sqrt(-trident1))/sqrt(-
trident1^3);
%             else
%                 C2_1 = 1/2;
%                 C3_1 = 1/6;
%             end
%             fprintf("im stuck\n")
%         end
%         % Now find other stuff (what is going ON with these variables maine)
%         f1 = 1 - (y1/r1_1);
%         g1 = A1*sqrt(y1/mew_sun);
%         g_dot1 = 1 - (y1/r2_1);
%         v1_vec1 = (r2_vec1 - (f1*r1_vec1))/g1; % Initial Vel
%         v2_vec1 = ((g_dot1*r2_vec1) - r1_vec1)/g1; % Final vel
%
%         % Algorithm for the long way (its the same, just new variable
%         % changes... I KNOW THIS SUCKS I'M PRESSED FOR TIME CUT ME SOME
%         % SLACK... It works though! (at least I think)
%         TOF2 = TOF1;
%         r1_vec2 = r1_vec1;
%         r2_vec2 = r2_vec1;
%         vel_earth2 = vel_earth;
%         vel_mars2 = vel_mars;
%         r1_2 = norm(r1_vec2);
%         r2_2 = norm(r2_vec2);
%         cos_deltaV2 = (dot(r1_vec2,r2_vec2))/(r1_2*r2_2);
%         DM2 = -1; % Given that the thang is short way
%         A2 = DM2*sqrt(r1_2*r2_2*(1+cos_deltaV2));
%
%         % Weird starter numbers I guess?
%         trident2 = 0;
%         C2_2 = 1/2;
%         C3_2 = 1/6;
%         trident_hp2 = 4*(pi^2);
%         trident_low2 = -4*(pi^2);
%         deltat2 = 0;
%
%         while(abs(TOF2-deltat2) >= tolerance)
%             y2 = r1_2 + r2_2 + (A2*((trident2*C3_2)-1))/sqrt(C2_2);
%             if(A2 > 0 && y2 <0)
%                 trident_low2 = trident_low2 + (pi/4);
%             end
%             x2 = sqrt(y2/C2_2);
%             deltat2 = (((x2^3)*C3_2)+(A2*sqrt(y2)))/sqrt(mew_sun);
%             if deltat2 < TOF2
%                 trident_low2 = trident2;
%             else
%                 trident_hp2 = trident2;
%             end
%             trident2 = (trident_hp2 + trident_low2)/2;
```

```
%             if trident2 > tolerance
%                 C2_2 = (1-cos(sqrt(trident2)))/trident2;
%                 C3_2 = (sqrt(trident2)-sin(sqrt(trident2)))/
sqrt(trident2^3);
%             elseif trident2 < -tolerance
%                 C2_2 = (1-cosh(sqrt(-trident2)))/trident2;
%                 C3_2 = (sinh(sqrt(-trident2))-sqrt(-trident2))/sqrt(-
trident2^3);
%             else
%                 C2_2 = 1/2;
%                 C3_2 = 1/6;
%             end
%             fprintf(int2str(trident2) + "\n")
%         end
%         % Now find other stuff (what is going ON with these variables maine)
%         f2 = 1 - (y2/r1_2);
%         g2 = A2*sqrt(y2/mew_sun);
%         g_dot2 = 1 - (y2/r2_2);
%         v1_vec2 = (r2_vec2 - (f2*r1_vec2))/g2;
%         v2_vec2 = ((g_dot2*r2_vec2) - r1_vec2)/g2;
%
%         % Now, do the stuff Prof. Barbee said so we can pick our correct
%         % V_infinites and stuff
%
%         % Fist case, short way
%         V_infinity_earth1 = v1_vec1 - vel_earth;
%         V_infinity_mars1 = v2_vec1 - vel_mars;
%         % Get the magnitude:
%         val1 = norm(V_infinity_earth1) + norm(V_infinity_mars1);
%
%         % Second case, long way
%         V_infinity_earth2 = v1_vec2 - vel_earth2;
%         V_infinity_mars2 = v2_vec2 - vel_mars2;
%         % Get the magnitude:
%         val2 = norm(V_infinity_earth2) + norm(V_infinity_mars2);
%
%         % now find which mag sum is smaller, and keep it:
%         if (val1 < val2)
%             C3_REAL= (norm(V_infinity_earth1))^2;
%             V_infinity_REAL = norm(V_infinity_mars1);
%         elseif (val2 < val1)
%             C3_REAL = (norm(V_infinity_earth2))^2;
%             V_infinity_REAL = norm(V_infinity_mars2);
%         else % for some reason if either conditional doesn't occur, just
 keep the short way
%             C3_REAL= (norm(V_infinity_earth1))^2;
%             V_infinity_REAL = norm(V_infinity_mars1);
%         end
%         V_infinity_matrix(i+1,ii+1) = V_infinity_REAL;
%         C3_martix(i+1,ii+1) = C3_REAL;
%
%     end
%
% end
```

```
figure
hold on
contour(TOF_matrix, "-k", DisplayName="TOF")
contour(V_infinity_matrix, "-r", DisplayName="V_i_n_f_i_n_i_t_y")
contour(C3_martix, "-b", DisplayName="C3")
```

```
% 2-body prop solver, pulled from ENAE301! Ah good times... good times.
function ydot = myodefun(t, y, mew, thrust)
    r_mag = norm(y(1:3));
    ydot(1,1) = y(4);
    ydot(2,1) = y(5);
    ydot(3,1) = y(6);

    % Get the unit vector of velocity vector
    unit_vec = [y(4);y(5);y(6)]/(norm([y(4);y(5);y(6)]));

    % Get the thrust vector
    thrust_vec = unit_vec*thrust;

    % Add it to accel
    ydot(4,1) = (-mew/r_mag^3)*y(1) + thrust_vec(1);
    ydot(5,1) = (-mew/r_mag^3)*y(2) + thrust_vec(2);
    ydot(6,1) = (-mew/r_mag^3)*y(3) + thrust_vec(3);
end
```



Published with MATLAB® R2022b