Table of Contents


```
%-------- HW 5 MATLAB code --------%
% Romeo Perlstein, section 0101 %

% IT'S A NEW WEEK - HOPEFULLY I CAN KEEP UP THE WORK AND NOT FALL BEHIND!
```

Q1

Find deltaV for Mercury to Jupiter transfer using conic sections

```
% Givens:
mew_mercury = 22031.868551; % km^3/s^2 - FROM JPL
mew_saturn = 126712764.1; % km^3/s^2 - FROM JPL
radius_planet_mercury = 2440.5; % km - from NASA fact sheet
r_craft_mercury = 400+radius_planet_mercury; % km
r_mercury = 57.909*10^6; % semi-major axis - from NASA fact sheet

radius_planet_saturn = 60268; % km - from NASA fact sheet
r_craft_saturn = 10000+radius_planet_saturn; % km
r_saturn = 1432.041*10^6; % semi-major axis - from NASA fact sheet

% since we are assuming circular orbits, we need to find the orbit
% velocity of both planets!
mew_sun = 132712*10^6; % from NASA fact sheet
v_mercury = sqrt(mew_sun/r_mercury);
v_saturn = sqrt(mew_sun/r_saturn);

% find velocities of orbits of spacecraft
v_initial_craft_mercury = sqrt(mew_mercury/r_craft_mercury);
v_final_craft_saturn = sqrt(mew_saturn/r_craft_saturn);

% Find the velocity to transfer from mercury to saturn
v_transfer_peri = sqrt(2*((mew_sun/r_mercury) - (mew_sun/(r_mercury
+r_saturn))));
v_transfer_apo = sqrt(2*((mew_sun/r_saturn) - (mew_sun/(r_mercury
+r_saturn))));
```

```
% Now, lets get the escape velocity from mercury
v_escape_mercury = v_transfer_peri - v_mercury;
v_hyperbola_peri_mercury = sqrt(2*((mew_mercury/r_craft_mercury) +
 ((v_escape_mercury^2)/2)));

% Now we can find the delta V to get to Saturn
deltaV1 = v_hyperbola_peri_mercury - v_initial_craft_mercury;

% Now do saturn
v_escape_saturn = v_saturn - v_transfer_apo;

% MAKING ASSUMPTION THAT HYPERBOLA PERIAPSIS IS SAME AS PARKING ORBIT
% PERIAPSIS, A. BECAUSE THE PROBLEM DOESN'T SAY WE CAN'T AND B. BECAUSE I
% WOULD NOT BE ABLE TO SUBMIT THE HW ON TIME
v_hyperbola_peri_saturn = sqrt(2*((mew_saturn/r_craft_saturn) +
 ((v_escape_saturn^2)/2)));
deltaV2 = v_hyperbola_peri_saturn - v_final_craft_saturn;

% now get the supplementary info
mass_mercury = .3301*10^24; % kg
mass_saturn = 568.32*10^24; % kg
mass_sun = 1998500*10^24; % kg
SOI_mercury = r_mercury*(mass_mercury/mass_sun)^(2/5); % km - Matches with
 Wikipedia!
SOI_saturn = r_saturn*(mass_saturn/mass_sun)^(2/5); % km - Matches with
 Wikipedia!
deltaV_total = deltaV1+deltaV2;

% Find TOF, assuming ellipse:
a = (r_mercury+r_saturn)/2;
e = 1-(r_mercury/a);
E = pi;
t = sqrt((a^3)/mew_sun)*(pi-0.9223*sin(pi)); % seconds!!!
```

Q2

do a bunch of stuff I don't have time to finish :/ given:

```
e2 = 1.2;
rp2 = 5380;
mew_mars = 0.042828 *10^6; % km^3/s^2 - from NASA fact sheet
% assuming circular orbit
r_mars = 227.956 * 10^6;
v_mars = sqrt(mew_sun/r_mars);

% find escape velocity and velocity at periapsis
v_escape_mars2 = sqrt(((e2-1)*mew_mars)/rp2); % from notes? I guess it works
v_hyperbola_peri_mars2 = sqrt(2*((mew_mars/rp2) + ((v_escape_mars2^2)/2))); %
 unused

% Get the hyperbola angle
Beta = acosd(1/e2); % get Beta angle
```

a

find v_initial, v_final, and then find deltaV

```
v_mars_vecA = [0;v_mars]; % Only acts in Y direction
v_escape_mars2_vec = [v_escape_mars2*sind(Beta);v_escape_mars2*cosd(Beta)]; %
 Get the vector components
v_initalA = v_mars_vecA + v_escape_mars2_vec; % Get the vector sum

v_escape_mars2_vec = [v_escape_mars2*sind(Beta);-v_escape_mars2*cosd(Beta)]; %
 Get the new escape vector
v_finalA = v_mars_vecA + v_escape_mars2_vec; % Get the vector sum

turn_angle = (180-(2*Beta))*(pi/180); % get the turn angle of the parabola
deltaV_A = sqrt(2*(v_escape_mars2^2)*cos(1-turn_angle)); % get the deltaV
 using the deltaV eq. from notes

% Now get the norms
v_initalA_mag = norm(v_initalA);
v_finalA_mag = norm(v_finalA);
```

b

find v_initial, v_final, and then find deltaV

```
mars_vec_angle = 30; % Given angle away from velocity vector

% Get velocity vectors
v_mars_vecB = [v_mars*sind(mars_vec_angle);v_mars*cosd(mars_vec_angle)];
v_escape_mars2_vec = [v_escape_mars2*sind(Beta);v_escape_mars2*cosd(Beta)];
v_initalB = v_mars_vecB + v_escape_mars2_vec; % Get vector sum

% get final stuff
v_escape_mars2_vec = [v_escape_mars2*sind(Beta);-v_escape_mars2*cosd(Beta)];
v_finalB = v_mars_vecB + v_escape_mars2_vec; % Get vector sum

% Get deltaV (turn angle does not change!)
deltaV_B = sqrt(2*(v_escape_mars2^2)*cos(1-turn_angle));

% Get magnitudes
v_initalB_mag = norm(v_initalB);
v_finalB_mag = norm(v_finalB);
```

c

do pretty much exactly the same thing as above, again

```
v_mars_vecC = [0; -v_mars];
v_escape_mars2_vec = [v_escape_mars2*sind(Beta);v_escape_mars2*cosd(Beta)];
v_initalC = v_mars_vecC + v_escape_mars2_vec;

v_escape_mars2_vec = [v_escape_mars2*sind(Beta);-v_escape_mars2*cosd(Beta)];
v_finalC = v_mars_vecC + v_escape_mars2_vec;
```

```
deltaV_C = sqrt(2*(v_escape_mars2^2)*cos(1-turn_angle));

v_initalC_mag = norm(v_initalC);
v_finalC_mag = norm(v_finalC);
```

Q3

solve keplers problamo

spacecraft 1

```
mew_earth = 0.39860*10^6;
a1 = 15000;
e1 = 0.4;
i1 = 60;
Omega1 = 45;
w1 = 0;
true_anom1 = 145;
deltaT = 3600; % seconds in an hour

% first, find E1_0 and mean motion
E1_0 = acos((e1+cosd(true_anom1))/(1+e1*cosd(true_anom1))*pi/180);
n = sqrt((a1^3)/mew_earth);

% now, solve the equation for E and get a constant
% Assuming k = 0 for simplicity? not sure how to solve for two unknowns
k = 0;
C1 = deltaT/n - 2*pi*k + (E1_0 - e1*sin(E1_0));
E1_guess = pi; % initial guess
fx1 = C1 - E1_guess + e1*sin(E1_guess); % begin iteration to solve for E
fx1_prime = -1 + e1*E1_guess*cos(E1_guess);
E1_prev = E1_guess - fx1/fx1_prime;
for ii=1:1:100 % 100 iterations should do the trick
    fx1 = C1 - E1_prev + e1*sin(E1_prev); % begin iteration to solve for E
    fx1_prime = -1 + e1*E1_prev*cos(E1_prev);
    E1_new = E1_prev - fx1/fx1_prime;
    E1_prev = E1_new;
end
E1 = E1_new;

% Now, find true anomaly
% through algebra, we find that:
true_anom1_final = acos((cos(E1)-e1)/(1-e1*cos(E1))); % rads
true_anom1_final_deg = true_anom1_final*180/pi; % convert to degrees


% now put it through the orbital elements converter
[pos_vec1_initial, vel_vec1_initial, spef_energy1_intial] =
 orbitalElementsToCart(a1, e1, i1, Omega1, w1, true_anom1, mew_earth, "deg");
[pos_vec1_final, vel_vec1_final, spef_energy1_final] =
 orbitalElementsToCart(a1, e1, i1, Omega1, w1, true_anom1_final_deg,
 mew_earth, "deg");
```

```
fprintf("Position and Velocity vectors for spacecraft_1 after 1 hour in flight
\n")
fprintf("Initial:\n")
pos_vec1_initial
vel_vec1_initial
fprintf("\nFinal:\n")
pos_vec1_final
vel_vec1_final
```

spacecraft 2

```
a2 = 21000;
e2 = 0.6;
i2 = 90;
Omega2 = 0;
w2 = 0;
true_anom2 = 35;

% first, find E1_0 and mean motion
E2_0 = acos((e2+cosd(true_anom2))/(1+e2*cosd(true_anom2))*pi/180);
n2 = sqrt((a2^3)/mew_earth);

% now, solve the equation for E and get a constant
% Assuming k = 0 for simplicity? not sure how to solve for two unknowns
k = 0;
C2 = deltaT/n2 - 2*pi*k + (E2_0 - e2*sin(E2_0));
E2_guess = pi; % initial guess
fx2 = C2 - E2_guess + e2*sin(E2_guess); % begin iteration to solve for E
fx2_prime = -1 + e2*E2_guess*cos(E2_guess);
E2_prev = E2_guess - fx2/fx2_prime;
for ii=1:1:100 % 100 iterations should do the trick
    fx2 = C2 - E2_prev + e2*sin(E2_prev); % begin iteration to solve for E
    fx2_prime = -1 + e2*E2_prev*cos(E2_prev);
    E2_new = E2_prev - fx2/fx2_prime;
    E2_prev = E2_new;
end
E2 = E2_new;

% Now, find true anomaly
% through algebra, we find that:
true_anom2_final = acos((cos(E2)-e2)/(1-e2*cos(E2))); % rads
true_anom2_final_deg = true_anom2_final*180/pi; % convert to degrees


% now put it through the orbital elements converter
[pos_vec2_initial, vel_vec2_initial, spef_energy2_intial] =
 orbitalElementsToCart(a2, e2, i2, Omega2, w2, true_anom2, mew_earth, "deg");
[pos_vec2_final, vel_vec2_final, spef_energy2_final] =
 orbitalElementsToCart(a2, e2, i2, Omega2, w2, true_anom2_final_deg,
 mew_earth, "deg");
```

```
fprintf("Position and Velocity vectors for spacecraft_2 after 1 hour in flight
\n")
fprintf("Initial:\n")
pos_vec2_initial
vel_vec2_initial
fprintf("\nFinal:\n")
pos_vec2_final
vel_vec2_final
```

Conclusion

While (I believe) the above algorithm works, converting it to it's own specific function would be beneficial for general use. However, this current algorithm makes several assumptions. For one, it assumes that the orbits will always be ellipses (or circular), since it does not account for the TOF equation for a parabola or hyperbola. This is something that would need to be added! As well, we are assuming that the true_anom is always less that 180 degrees, and thus are not doing a half-plane check. finally and most important, we are only considering the possiblity that our predictions are relative to one orbit, as in, we are not attempting to solve if the craft makes multiple orbits in the given time (1 hour), rather just where it is relative to the orbit. In order to figure out how many orbits the craft has made as well, we would need to include the K parameter into the equation, rather than just setting it to zero. There is much to be desired for, which I assume is why it's called "Keplers problem" and not "Keplers solution" !!! - Romeo

```
Position and Velocity vectors for spacecraft_1 after 1 hour in flight
Initial:

pos_vec1_initial =

   1.0e+04 *

   -1.8456
    1.8456
         0


vel_vec1_initial =

   -0.6142
    0.6142
         0


Final:

pos_vec1_final =

   1.0e+04 *

   -1.8287
    1.8287
         0


vel_vec1_final =
```

```
    0.7975
   -0.7975
         0
```

```
Position and Velocity vectors for spacecraft_2 after 1 hour in flight
Initial:
```

```
pos_vec2_initial =

   1.0e+03 *

    7.3815
    5.1686
         0


vel_vec2_initial =

   -3.1236
    7.7285
         0


Final:

pos_vec2_final =

   1.0e+04 *

   -2.4785
    1.3683
         0


vel_vec2_final =

   -2.6320
   -1.5001
         0
```

Published with MATLAB® R2022b