```
%Gustavo Grinsteins
%ASEN 5050
%HW6
%Solving lambert's equation v 2

function [DepVinf,ArrivVinf] = EllOrbitLambertEqSolve2(R1,R2,V1,V2,TOF,mu,less180)
    r1 = norm(R1);
    r2 = norm(R2);
    %STEP 1: Find the transfer angle (theta<180))
    if less180
        delta_ThetaStar = abs(acos(dot(R1,R2)/(r1*r2)));
    end
    %STEP 2: Calculate Geometric quantities
    c = sqrt(r1^2+r2^2-2*(r1*r2)*cos(delta_ThetaStar));
    s = (1/2)*(r1+r2+c);
    %STEP 3: Determine if the TOF is for an ellipse
    TOFp = (1/3)*sqrt(2/mu)*((s^(3/2))+((s-c)^(3/2)));
    if TOF > TOFp
        fprintf('The transfer orbit is elliptical \n')
        %STEP 4: Determine correct alpha and beta
        a_m = (s/2);
        n_m = sqrt(mu/(a_m^3));
        alpha_m = pi;
        beta_m_0 = 2*asin(sqrt((s-c)/(s)));
        less180 = true;
        if less180
            beta_m = beta_m_0;%(theta<180)
        else
            beta_m = -beta_m_0;%(theta>180)
        end
        TOFmin = (1/n_m)*((alpha_m-beta_m)-(sin(alpha_m)-sin(beta_m)));
        %STEP 5: Usin Fsolve
        %Initial Guess for a
        a_initial = a_m+150;%how to defend delta_a?
        %Calc alpha boolean
        if TOF > TOFmin
            greatThanTOFmin = true;
        else
            greatThanTOFmin = false;
        end
        diff = 1000; %initial value for stopping condition
        diff2 = inf; %initial value for second stopping condition
        a_new = 0;
        iterations = 0;
        tolerance = 10^-5; %5 digit accuracy is desired
        while diff > tolerance %Convergence stopping condition
            %implement Fsolve function
            %define the anonymous function handle
            options = optimoptions('fsolve','Display','off');
            a_bef = a_new;
            a_new = fsolve(@(a)LambertEqt(mu,a,s,c,TOF,less180,greatThanTOFmin),↙
a_initial,options);
            %Recalculate values
            n_new = sqrt(mu/(a_new^3));
            alpha_0 = 2*asin(sqrt((s)/(2*a_new)));
            beta_0 = 2*asin(sqrt((s-c)/(2*a_new)));
            if less180
                beta = beta_0;
```

```
            else
                beta = -beta_0;
            end
            if TOF > TOFmin
                alpha = 2*pi - alpha_0;
            else
                alpha = alpha_0;
            end
            TOF_new = (1/n_new)*((alpha-beta)-(sin(alpha)-sin(beta)));
            diff = abs(TOF_new-TOF);
            %Divergence stopping condition
            if diff > diff2
                fprintf('Calculations started Diverging - Stopping iterations \n')
                a_new = a_bef;
                break
            end
            %Too many Iterations stopping condition
            if iterations > 300
                fprintf('Too many iterations reached, adapt your algorithm for the↙
problem \n')
                break
            end
            diff2 = diff;
            fprintf('diff = %0.12f \n',diff)
            a_initial = a_new + 150;
            iterations = iterations +1;
        end
        %calculating v infinity
        at = a_new;
        alpha_0 = 2*asin(sqrt((s)/(2*at)));
        beta_0 = 2*asin(sqrt((s-c)/(2*at)));
        if less180
            beta = beta_0;
        else
            beta = -beta_0;
        end
        if TOF > TOFmin
            alpha = 2*pi - alpha_0;
        else
            alpha = alpha_0;
        end
        p =((4*at*(s-r1)*(s-r2))/(c^2))*sin((alpha+beta)/(2)).^2;
        %Calculating transfer speeds using f and G functions
        f = 1- (r2/p)*(1-cos(delta_ThetaStar));
        g = (r2*r1*sin(delta_ThetaStar))/(sqrt(mu*p));
        f_dot = sqrt(mu/p)*tan(delta_ThetaStar/2)*(((1-cos(delta_ThetaStar))/(p))-(1/r2)↙
-(1/r1));
        g_dot = 1-(r1/p)*(1-cos(delta_ThetaStar));
        V1_f = (1/g)*(R2-f*R1);
        DepVinf = norm(V1_f-V1);%Earth Departing Vinf
        V2_i = (f_dot)*R1+g_dot*V1_f;
        ArrivVinf = norm(V2_i-V2);%Mars Arriving Vinf

    else
        fprintf('The transfer orbit is not elliptical \n')
        DepVinf = NaN;%Earth Departing Vinf
        ArrivVinf = NaN;%Mars Arriving Vinf
    end
```

```
end
```