```
function out = problem2_function(mu, R1, V1, R2, V2, TOF, delta_a,
greater_than_180)
    % Solve lambert's problem using fsolve using only elliptical arcs and
    % output v_infinity for porkchop plots
    % Inputs - gravitational parameter of central body,
    %          initial pos, vel, final pos, vel, Time of Flight,
    %          initial guess delta a, whether choice is greater/lower than
    %          180 deg
    % Output - v_inf_earth and v_inf_mars in km/s

    % Compute norms of initial and final positions
    r1 = norm(R1);
    r2 = norm(R2);

    % Step 1 - Transfer angle
    delta_theta_star = acos(dot(R1, R2) / (r1*r2));

    % If greater than 180, subtract original value by 360 deg to get a
    % delta theta star that's > 180 deg
    if greater_than_180
        delta_theta_star = 2*pi - delta_theta_star;
    end

    % Step 2 - calculate c and s
    c = sqrt(r1^2 + r2^2 - 2*r1*r2*cos(delta_theta_star));
    s = 0.5 * (r1 + r2 + c);

    % Step 3 is figuring out if arc is elliptical or hyperbolic but since
    % this only uses elliptical arcs, that step is eliminated.

    % Step 4 - Min energy transfer arc
    am = s/2;
    alpha_m = pi;
    nm = sqrt(mu/am^3);
    beta_m_0 = 2 * asin(sqrt((s-c)/s));
    beta_m = beta_m_0;
    if greater_than_180
        beta_m = -beta_m_0;
    end
    TOF_min = 1/nm * (alpha_m - beta_m - (sin(alpha_m) - sin(beta_m)));

    % Step 5 - iteratively solve for a
    a_initial_guess = am + delta_a;

    % Get function where the variable is a
    fun = @(a)lamberts_eqn_elliptical(a, s, c, TOF, mu, TOF_min,
greater_than_180);

    % Use fsolve to output a where function goes to 0
    options = optimoptions('fsolve', 'Display', 'off');
    a = fsolve(fun, a_initial_guess, options);
```

```
    % Compute final alpha, beta values to compare
    alpha_0_final = 2 * asin(sqrt(s/(2*a)));
    beta_0_final = 2 * asin(sqrt((s-c)/(2*a)));
    alpha_beta_final = lamberts_problem_alpha_beta_logic(TOF, TOF_min,
greater_than_180, alpha_0_final, beta_0_final);
    alpha_final = alpha_beta_final(1);
    beta_final = alpha_beta_final(2);
    n_final = sqrt(mu/a^3);

    % Step 6
    TOF_final = 1/n_final * (alpha_final - beta_final - (sin(alpha_final) -
sin(beta_final)));
    if abs(TOF_final - TOF) > 1e-4
        disp("TOFs don't match and the abs error is " + abs(TOF_final-TOF) +
" seconds.")
    end

    % Step 7 - find p and e
    p = (4*a*(s-r1)*(s-r2))/(c^2) * (sin((alpha_final + beta_final)/2))^2;
    e = sqrt(1 - p/a);

    % Step 8 - find true anomaly at each location
    theta_star_1_p = abs(acos(1/e * (p/r1 - 1)));
    theta_star_2_p = abs(acos(1/e * (p/r2 - 1)));
    theta_star_1_n = -theta_star_1_p;
    theta_star_2_n = -theta_star_2_p;

    % Test matrix
    % First column = difference between 2 and 1
    % Second column = sign of theta_star_2
    % Third column = sign of theta_star_1
    test_ones = ones(6, 2);
    test = [zeros(6, 1), test_ones];
    test(1, 1) = theta_star_2_p - theta_star_1_p;
    test(2, 1) = theta_star_2_p - theta_star_1_n;
    test(2, 3) = -1;
    test(3, 1) = theta_star_2_n - theta_star_1_p;
    test(3, 2) = -1;
    test(4, 1) = theta_star_2_n - theta_star_1_n;
    test(4, 2) = -1;
    test(4, 3) = -1;
    test(5, 1) = 2*pi - theta_star_2_p - theta_star_1_p;
    test(5, 2) = -1;
    test(6, 1) = 2*pi - theta_star_2_p - theta_star_1_n;
    test(6, 2) = -1;
    test(6, 3) = -1;

    % Parse through matrix to get correct transfer angle
    for i = 1:length(test)
        if abs(test(i) - delta_theta_star) < 1e-6
            theta_star_1 = theta_star_1_p * test(i, 3);
            theta_star_2 = theta_star_2_p * test(i, 2);
            break
        else
```

```
            if i == length(test)
                disp("Needs manual interference in delta_theta_star calc")
            end
        end

    end

    % Step 9 - find f, g to compute v1f, v2i
    f_g_fdot_gdot = fg_out(R1, R2, delta_theta_star, p, mu);
    f = f_g_fdot_gdot(1);
    g = f_g_fdot_gdot(2);
    fdot = f_g_fdot_gdot(3);
    gdot = f_g_fdot_gdot(4);

    v_1_f = (R2 - f*R1)./g;
    v_2_i = fdot*R1 + gdot*v_1_f;

    % Step 10 - v_inf for earth and mars
    v_inf_earth = norm(v_1_f - V1);
    v_inf_mars = norm(v_2_i - V2);

    out = [v_inf_earth, v_inf_mars];
end


function fx = lamberts_eqn_elliptical(a, s, c, TOF, mu, TOF_min,
greater_than_180)
    % Output function to solve for fsolve
    % Inputs:
    %       a - Semi-major axis of transfer conic [km]
    %       s - semi-perimeter of transfer triangle [km]
    %       c - chord length of transfer triangle [km]
    %       TOF - time of flight of transfer [s]
    %       mu - gravitational parameter of central body [km^3/s^2\
    %       TOF_min - TOF for minimum energy transfer [s]
    %       greater_than_180 - bool that's true if chosen arc is >180 deg
    % Output:
    %       fx - function that's dependent on a to pass to fsolve

    % Mean anomaly for transfer ellipse
    n = sqrt(mu/a^3);

    % Calculate principal alpha, beta
    alpha_0 = 2 * asin(sqrt(s/(2*a)));
    beta_0 = 2 * asin(sqrt((s-c)/(2*a)));

    % Get correct alpha, beta dependent on TOF and (choice of)
    % greater_than_180 bool
    alpha_beta = lamberts_problem_alpha_beta_logic(TOF, TOF_min,
greater_than_180, alpha_0, beta_0);
    alpha = alpha_beta(1);
    beta = alpha_beta(2);

    % Output function
```

```
    fx = TOF - (alpha - beta - (sin(alpha) - sin(beta)))/n;
end
```

```
Not enough input arguments.

Error in problem2_function (line 11)
    r1 = norm(R1);
```

Published with MATLAB® R2024a