Table of Contents


```
function transfer = solveLambertsProblem(initialState, finalState, TOF,
lt180, mu)
% Function that solves Lambert's Problem for a given initial and final
% state subject to a desired flight time and transfer angle either less or
% greater than 180 degrees.
%   Inputs:
%       - initialState: Initial state of the spacecraft
%           - [X; Y; Z; Vx; Vy; Vz]
%       - finalState: Final state of the spacecraft
%           - [X; Y; Z; Vx; Vy; Vz]
%       - TOF: Desired TOF, calculated from the initial and final epoch in
%              SECONDS (e.g. epoch of planet 2 at arrival minus epoch of
%              planet 1 at departure)
%       - lt180: Whether the desired transfer angle is less than or greater
%                than 180 degrees
%           - lt180 = 0: transfer angle greater than 180 degrees
%           - lt180 = 1: transfer angle less than 180 degrees
%       - mu: Gravitational parameter of the body governing the 2BP
%             dynamics of the transfer
%           - This is generally the sun
%   Outputs:
%       - transfer: Transfer output structure with the following fields:
%           - a: Calculated transfer semi-major axis in km
%           - e: Calculated transfer eccentricity (unitless)
%           - TOF_calc: Calculated transfer TOF in seconds
%           - dTA_deg: Transfer angle in degrees
%           - TA1_deg: True anomaly at the start of the transfer in degrees
%           - TA2_deg: True anomaly at the end of the transfer in degrees
%           - Vt1: Initial velocity of transfer in km/s
%           - Vt2: Final velocity of transfer in km/s
%           - dV1: Delta V vector to enter the transfer in km/s
%           - dV1_mag: Magnitude of delta V to enter the transfer in km/s
%           - dV2: Delta V vector to exit the transfer in km/s
%           - dV2_mag: Magnitude of delta V to exit the transfer in km/s
%           - dV_mag_total: Total delta V required to execute transfer in
%                           km/s
%           - Vinfinity_1: V_infinity with respect to planet 1 in km/s
%           - Vinfinity_2: V_infinity with respect to planet 2 in km/s
%           - orbElems: subset of orbital elements of this transfer, saved
%                       as a struct with fields inc, RAAN, and argPeri in
%                       radians
%           - type: Type of transfer (elliptical or hyperbolic)
%
%   By: Ian Faber, 10/20/2024
%

% Extract initial and final states
```

```
R1 = initialState(1:3);
V1 = initialState(4:6);
R2 = finalState(1:3);
V2 = finalState(4:6);

% Find transfer angle
dTA_deg = acosd(dot(R1,R2)/(norm(R1)*norm(R2)));
    % Want dTA to be within [0, 360] deg
dTA_1_deg = abs(dTA_deg);
dTA_2_deg = 360 - dTA_1_deg;

if lt180 % Extract transfer angle that's less than 180 degrees
    if dTA_1_deg < 180
        dTA_deg = dTA_1_deg;
    else
        dTA_deg = dTA_2_deg;
    end
else % Extract transfer angle that's greater than 180 degrees
    if dTA_1_deg > 180
        dTA_deg = dTA_1_deg;
    else
        dTA_deg = dTA_2_deg;
    end
end

% Calculate space triangle
r1 = norm(R1);
r2 = norm(R2);
c = sqrt(r1^2 + r2^2 - 2*r1*r2*cosd(dTA_deg));
s = 0.5*(r1 + r2 + c);

% Calculate parabolic TOF
if lt180
    TOF_p = (1/3)*sqrt(2/mu)*(s^(3/2) - (s-c)^(3/2));
else
    TOF_p = (1/3)*sqrt(2/mu)*(s^(3/2) + (s-c)^(3/2));
end

if TOF_p < TOF % Elliptical transfer needed
    ellipse = true;
    type = "Elliptical";
else % Hyperbolic transfer needed
    ellipse = false;
    type = "Hyperbolic";
end

% Calculate minimum energy transfer TOF
amin = s/2;
nmin = sqrt(mu/(amin)^3);
alphamin = pi;
betamin0 = 2*asin(sqrt((s-c)/s));
if dTA_deg < 180
    betamin = betamin0;
else
```

```
    betamin = -betamin0;
end
TOFmin = (1/nmin)*((alphamin - betamin) - (sin(alphamin) - sin(betamin)));

if TOF < TOFmin
    shortTOF = 1;
else
    shortTOF = 0;
end

% Solve for a of transfer
a = solveLambertsEq(mu, s, c, TOF, shortTOF, lt180, ellipse);

% Recreate alpha and beta
if ellipse
    alpha0 = 2*asin(sqrt(s/(2*a)));
    beta0 = 2*asin(sqrt((s-c)/(2*a)));
else
    alpha0 = 2*asinh(sqrt(s/(2*abs(a))));
    beta0 = 2*asinh(sqrt((s-c)/(2*abs(a))));
end

if shortTOF
    alpha = alpha0;
else
    alpha = 2*pi - alpha0;
end

if lt180
    beta = beta0;
else
    beta = -beta0;
end

n = sqrt(mu/(a^3));

% Calculate resulting TOF
if ellipse
    TOF_calc = (1/n)*((alpha - beta) - (sin(alpha) - sin(beta)));
else
    if lt180
        TOF_calc = (1/n)*(sinh(alpha) - alpha - (sinh(beta) - beta));
    else
        TOF_calc = (1/n)*(sinh(alpha) - alpha + (sinh(beta) - beta));
    end
end

% Calculate eccentricity
p = ((4*a*(s-r1)*(s-r2))/(c^2))*(sin(0.5*(alpha+beta)))^2;

e = sqrt(1-(p/a));

% Calculate true anomaly at start and end of transfer
TA1_deg_init = acosd((1/e)*(p/r1 - 1));
```

```
TA2_deg_init = acosd((1/e)*(p/r2 - 1));

% Check for correct TA combination
sit = 1;
minDiff = 1000; % Dummy variable for storing the last minimum difference
while true
    switch sit
        case 1
            TA1_deg = TA1_deg_init;
            TA2_deg = TA2_deg_init;
        case 2
            TA1_deg = -TA1_deg_init;
            TA2_deg = TA2_deg_init;
        case 3
            TA1_deg = TA1_deg_init;
            TA2_deg = -TA2_deg_init;
        case 4
            TA1_deg = -TA1_deg_init;
            TA2_deg = -TA2_deg_init;
        case 5
            TA1_deg = 360 - TA1_deg_init;
            TA2_deg = TA2_deg_init;
        case 6
            TA1_deg = -(360 - TA1_deg_init);
            TA2_deg = TA2_deg_init;
        case 7
            TA1_deg = 360 - TA1_deg_init;
            TA2_deg = -TA2_deg_init;
        case 8
            TA1_deg = -(360 - TA1_deg_init);
            TA2_deg = -TA2_deg_init;
        case 9
            TA1_deg = TA1_deg_init;
            TA2_deg = 360 - TA2_deg_init;
        case 10
            TA1_deg = -TA1_deg_init;
            TA2_deg = 360 - TA2_deg_init;
        case 11
            TA1_deg = TA1_deg_init;
            TA2_deg = -(360 - TA2_deg_init);
        case 12
            TA1_deg = -TA1_deg_init;
            TA2_deg = -(360 - TA2_deg_init);
        case 13
            TA1_deg = 360 - TA1_deg_init;
            TA2_deg = 360 - TA2_deg_init;
        case 14
            TA1_deg = -(360 - TA1_deg_init);
            TA2_deg = 360 - TA2_deg_init;
        case 15
            TA1_deg = 360 - TA1_deg_init;
            TA2_deg = -(360 - TA2_deg_init);
        case 16
            TA1_deg = -(360 - TA1_deg_init);
```

```
            TA2_deg = -(360 - TA2_deg_init);
    end
    dTA_check = TA2_deg - TA1_deg;

    if abs(dTA_check - dTA_deg) < minDiff
        minDiff = abs(dTA_check - dTA_deg);
        tempTA1 = TA1_deg;
        tempTA2 = TA2_deg;
    end

    if sit < 16 % Only test these 16 combinations
        sit = sit + 1; % Iterate to the next check
    else
        break % Tested all combinations!
    end
end
TA1_deg = tempTA1;
TA2_deg = tempTA2;

% Calculate velocities at start and end of transfer
f = 1 - (r2/p)*(1-cosd(dTA_deg));
g = ((r2*r1)/sqrt(mu*p))*sind(dTA_deg);
fDot = sqrt(mu/p)*tand(dTA_deg/2)*(((1-cosd(dTA_deg))/p) - (1/r2) - (1/r1));
gDot = 1 - (r1/p)*(1-cosd(dTA_deg));

Vt1 = (1/g)*(R2 - f*R1);
Vt2 = fDot*R1 + gDot*Vt1;

% Calculate delta V's
dV1 = Vt1 - V1;
dV1_mag = norm(dV1);
dV2 = V2 - Vt2;
dV2_mag = norm(dV2);
dV_mag_total = dV1_mag + dV2_mag;

% Calculate Vinfinities: Vinfinity = norm(V_s/c - V_planet)
Vinfinity_1 = norm(Vt1 - V1);
Vinfinity_2 = norm(Vt2 - V2);

% Calculate orbital elements of transfer
vt1 = norm(Vt1);
r1 = norm(R1);
h_vec = cross(R1,Vt1);
e_vec = ((vt1^2 - (mu/r1))*R1 - dot(R1, Vt1)*Vt1)/mu;
n_vec = cross([0;0;1],h_vec);

orbElems.inc = acos(dot(h_vec, [0;0;1])/norm(h_vec));
orbElems.RAAN = acos(dot(n_vec, [1;0;0])/norm(n_vec))*sign(dot(n_vec,
[0;1;0]));
orbElems.argPeri = acos(dot(n_vec,e_vec)/
(norm(n_vec)*norm(e_vec)))*sign(dot(e_vec, [0;0;1]));

% Assign outputs
transfer.a = a;
```

```
transfer.e = e;
transfer.TOF_calc = TOF_calc;
transfer.dTA_deg = dTA_deg;
transfer.TA1_deg = TA1_deg;
transfer.TA2_deg = TA2_deg;
transfer.Vt1 = Vt1;
transfer.Vt2 = Vt2;
transfer.dV1 = dV1;
transfer.dV1_mag = dV1_mag;
transfer.dV2 = dV2;
transfer.dV2_mag = dV2_mag;
transfer.dV_mag_total = dV_mag_total;
transfer.Vinfinity_1 = Vinfinity_1;
transfer.Vinfinity_2 = Vinfinity_2;
transfer.orbElems = orbElems;
transfer.type = type;

end
```

Published with MATLAB® R2023b