```
function [vi, vf] = glambert(cbmu, sv1, sv2, tof, nrev)

% Gooding's solution of Lambert's problem

% input

%  cbmu = central body gravitational constant
%  sv1  = initial 6-element state vector (position + velocity)
%  sv2  = final 6-element state vector (position + velocity)
%  tof  = time of flight (+ posigrade, - retrograde)
%  nrev = number of full revolutions
%         (positive for long period orbit,
%          negative for short period orbit)

% output

%  vi = initial velocity vector of the transfer orbit
%  vf = final velocity vector of the transfer orbit

% References

%  R. H. Gooding, Technical Report 88027
%  On the Solution of Lambert's Orbital Boundary-Value Problem,
%  Royal Aerospace Establishment, April 1988

%  R. H. Gooding, Technical Memo SPACE 378
%  A Procedure for the Solution of Lambert's Orbital Boundary-Value Problem
%  Royal Aerospace Establishment, March 1991

% Orbital Mechanics with Matlab

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

r1mag = norm(sv1(1:3));

r2mag = norm(sv2(1:3));

ur1xv1 = cross(sv1(1:3), sv1(4:6));

ur1xv1 = ur1xv1 / norm(ur1xv1);

ux1 = sv1(1:3) / r1mag;

ux2 = sv2(1:3) / r2mag;

uz1 = cross(ux1, ux2);

uz1 = uz1 / norm(uz1);

% calculate the minimum transfer angle (radians)

theta = dot(ux1, ux2);
```

```
if (theta > 1.0)
    theta = 1.0;
end

if (theta < -1.0)
    theta = -1.0;
end

theta = acos(theta);

% calculate the angle between the orbit normal of the initial orbit
% and the fundamental reference plane

angle_to_on = dot(ur1xv1, uz1);

if (angle_to_on > 1.0)
    angle_to_on = 1.0;
end

if (angle_to_on < -1.0)
    angle_to_on = -1.0;
end

angle_to_on = acos(angle_to_on);

% if angle to orbit normal is greater than 90 degrees
% and posigrade orbit, then flip the orbit normal
% and the transfer angle

if ((angle_to_on > 0.5 * pi) && (tof > 0.0))
    theta = 2.0 * pi - theta;

    uz1 = -uz1;
end

if ((angle_to_on < 0.5 * pi) && (tof < 0.0))
    theta = 2.0 * pi - theta;

    uz1 = -uz1;
end

uz2 = uz1;

uy1 = cross(uz1, ux1);

uy1 = uy1 / norm(uy1);

uy2 = cross(uz2, ux2);

uy2 = uy2 / norm(uy2);

theta = theta + 2.0 * pi * abs(nrev);
```

```
[vr11, vr12, vr21, vr22, vt11, vt12, vt21, vt22, n] = vlamb(cbmu, r1mag,
 r2mag, theta, tof);

if (nrev > 0)
    if (n == -1)
        disp ('no tminium')

        vi = 0.0;

        vf = 0.0;
    elseif (n == 0)
        disp ('no solution time')

        vi = 0.0;

        vf = 0.0;
    elseif (n == 1)
        disp ('one solution')
    else
        disp ('two solutions')
    end
end

% compute transfer orbit initial and final velocity vectors

if ((nrev > 0) && (n > 1))
    vi = vr21 * ux1 + vt21 * uy1;

    vf = vr22 * ux2 + vt22 * uy2;
else
    vi = vr11 * ux1 + vt11 * uy1;

    vf = vr12 * ux2 + vt12 * uy2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [dt, d2t, d3t, t] = tlamb(m, q, qsqfm1, x, n)

% Gooding lambert support function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

lm1 = false;

l1 = false;

l2 = false;

l3 = false;

sw = 0.4;
```

```
lm1 = n == -1;

l1 = n >= 1;

l2 = n >= 2;

l3 = n == 3;

qsq = q * q;

xsq = x * x;

u = (1.0 - x) * (1.0 + x);

% needed if series, and otherwise useful when z = 0

if (~lm1)
    dt = 0.0;

    d2t = 0.0;

    d3t = 0.0;
end

if (lm1 || m > 0 || x < 0.0 || abs(u) > sw)
    % direct computation, not series

    y = sqrt(abs(u));

    z = sqrt(qsqfm1 + qsq * xsq);

    qx = q * x;

    if (qx <= 0.0)
        a = z - qx;

        b = q * z - x;
    end

    if ((qx < 0) && lm1)
        aa = qsqfm1 / a;

        bb = qsqfm1 * (qsq * u - xsq) / b;
    end

    if (qx == 0.0 && lm1 || qx > 0.0)
        aa = z + qx;

        bb = q * z + x;
    end

    if (qx > 0)
        a = qsqfm1 / aa;
```

```
    b = qsqfm1 * (qsq * u - xsq) / bb;
end

if (lm1)
    dt = b;

    d2t = bb;

    d3t = aa;
else
    if (qx * u >= 0.0)
        g = x * z + q * u;
    else
        g = (xsq - qsq * u) / (x * z - q * u);
    end

    f = a * y;

    if (x <= 1)
        t = m * pi + atan2(f, g);
    else
        if (f > sw)
            t = log(f + g);
        else
            fg1 = f / (g + 1.0);

            term = 2.0 * fg1;

            fg1sq = fg1 * fg1;

            t = term;

            twoi1 = 1.0;

            told = 0.0;

            while (t ~= told)
                twoi1 = twoi1 + 2.0;

                term = term * fg1sq;

                told = t;

                t = t + term / twoi1;
            end
        end
    end

    t = 2.0 * (t / y + b) / u;

    if (l1 && z ~= 0.0)
        qz = q / z;

        qz2 = qz * qz;
```

```
            qz = qz * qz2;

            dt = (3.0 * x * t - 4.0 * (a + qx * qsqfm1) / z) / u;

            if (l2)
                d2t = (3.0 * t + 5.0 * x * dt + 4.0 * qz * qsqfm1) / u;
            end

            if (l3)
                d3t = (8.0 * dt + 7.0 * x * d2t ...
                    - 12.0 * qz * qz2 * x * qsqfm1) / u;
            end
        end
    end
else
    % compute by series

    u0i = 1.0;

    if (l1)
        u1i = 1.0;
    end

    if (l2)
        u2i = 1.0;
    end

    if (l3)
        u3i = 1.0;
    end

    term = 4.0;

    tq = q * qsqfm1;

    if (q < 0.5)
        tqsum = 1.0 - q * qsq;
    end

    if (q >= 0.5)
        tqsum = (1.0 / (1.0 + q) + q) * qsqfm1;
    end

    ttmold = term / 3.0;

    t = ttmold * tqsum;

    % start of loop

    icounter = 0;

    while (icounter < n || t ~= told)
        icounter = icounter + 1;
```

```
    p = icounter;

    u0i = u0i * u;

    if (l1 && icounter > 1)
        u1i = u1i * u;
    end

    if (l2 && icounter > 2)
        u2i = u2i * u;
    end

    if (l3 && icounter > 3)
        u3i = u3i * u;
    end

    term = term * (p - 0.5) / p;

    tq = tq * qsq;

    tqsum = tqsum + tq;

    told = t;

    tterm = term / (2.0 * p + 3.0);

    tqterm = tterm * tqsum;

    t = t - u0i * ((1.5 * p + 0.25) * tqterm / (p * p - 0.25) ...
        - ttmold * tq);

    ttmold = tterm;

    tqterm = tqterm * p;

    if (l1)
        dt = dt + tqterm * u1i;
    end

    if (l2)
        d2t = d2t + tqterm * u2i * (p - 1.0);
    end

    if (l3)
        d3t = d3t + tqterm * u3i * (p - 1.0) * (p - 2.0);
    end

end

if (l3)
    d3t = 8.0 * x * (1.5 * d2t - xsq * d3t);
end
```

```
    if(l2)
        d2t = 2.0 * (2.0 * xsq * d2t - dt);
    end

    if (l1)
        dt = -2.0 * x * dt;
    end

    t = t / xsq;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

function [vr11, vr12, vr21, vr22, vt11, vt12, vt21, vt22, n] = vlamb(cbmu, r1,
 r2, thr2, tdelt)

% Gooding lambert support function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vr21 = 0;

vr22 = 0;

vt21 = 0;

vt22 = 0;

m = 0;

while (thr2 > (2.0 * pi))
    thr2 = thr2 - 2.0 * pi;

    m = m + 1;
end

thr2 = thr2 / 2.0;

dr = r1 - r2;

r1r2 = r1 * r2;

r1r2th = 4.0 * r1r2 * sin(thr2)^2;

csq = dr^2 + r1r2th;

c = sqrt(csq);

s = (r1 + r2 + c) / 2.0;

gms = sqrt(cbmu * s / 2.0);
```

```
qsqfm1 = c/s;

q = sqrt(r1 * r2) * cos(thr2) / s;

if (c ~= 0.0)
    rho = dr / c;

    sig = r1r2th /csq;
else
    rho = 0.0;

    sig = 1.0;
end

t = 4.0 * gms * tdelt / s^2;

[x1, x2, n] = xlamb(m, q, qsqfm1, t);

% proceed for single solution, or a pair

for i = 1:1:n
    if (i == 1)
        x = x1;
    else
        x = x2;
    end

    [qzminx, qzplx, zplqx] = tlamb(m, q, qsqfm1, x, -1);

    vt2 = gms * zplqx * sqrt(sig);

    vr1 = gms * (qzminx - qzplx * rho) / r1;

    vt1 = vt2 / r1;

    vr2 = -gms * (qzminx + qzplx * rho) / r2;

    vt2 = vt2 / r2;

    if (i == 1)
        vr11 = vr1;

        vt11 = vt1;

        vr12 = vr2;

        vt12 = vt2;
    else
        vr21 = vr1;

        vt21 = vt1;

        vr22 = vr2;
```

```
        vt22 = vt2;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x, xpl, n] = xlamb(m, q, qsqfm1, tin)

% Gooding lambert support function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tol = 3.0e-7;

xpl = 0.0;

c0 = 1.7;

c1 = 0.5;

c2 = 0.03;

c3 = 0.15;

c41 = 1.0;

c42 = 0.24;

termflag = false;

thr2 = atan2(qsqfm1, 2.0 * q) / pi;

if (m == 0)
    % single rev starter from t (at x = 0) & bilinear (usually)

    n = 1;

    [dt, d2t, d3t, t0] = tlamb(m, q, qsqfm1, 0.0, 0.0);

    tdiff = tin - t0;

    if (tdiff <= 0.0)
        x = t0 * tdiff / (-4.0 * tin);
    else
        x = -tdiff / (tdiff + 4.0);

        w = x + c0 * sqrt(2.0 * (1.0 - thr2));

        if (w < 0.0)
            x = x - sqrt(d8rt(-w)) * (x + sqrt(tdiff / (tdiff + 1.5 * t0)));
        end
```

```
        w = 4.0 /(4.0 + tdiff);

        x = x * (1.0 + x * (c1 * w - c2 * x * sqrt(w)));
    end
else
    % with multirevs, first get t(min) as basis for starter

    xm = 1.0 / (1.5 * (m + 0.5) * pi);

    if (thr2 < 0.5)
        xm = d8rt(2.0 * thr2) * xm;
    end

    if (thr2 > 0.5)
        xm = (2.0 - d8rt(2.0 - 2.0 * thr2)) * xm;
    end;

    % starter for tmin

    for i = 1:1:12
        [dt, d2t, d3t, tmin] = tlamb(m, q, qsqfm1, xm, 3);

        if (d2t == 0.0)
            solnflag = true;
            break
        end

        xmold = xm;

        xm = xm - dt * d2t / (d2t * d2t - dt * d3t / 2.0);

        xtest = abs(xmold / xm - 1.0);

        if (xtest <= tol)
            solnflag = true;
            break
        end
    end

    % break off & exit if tmin not located - should never happen

    % now proceed from t(min) to full starter

    if (solnflag)
        tdiffm = tin - tmin;

        if (tdiffm < 0.0)
            n = 0;

            termflag = true;

            % exit if no solution with this m
        elseif (tdiffm == 0)
            x = xm;
```

```
            n = 1;

            termflag = true;

            % exit if unique solution already from x(tmin)
        else
            n = 3;

            if (d2t == 0)
                d2t = 6.0 * m * pi;
            end

            x = sqrt(tdiffm / (d2t / 2.0 + tdiffm / (1.0 - xm)^2));

            w = xm + x;

            w = w * 4.0 / (4.0 + tdiffm) + (1.0 - w)^2;

            x = x * (1.0 - (1.0 + m + c41 * (thr2 - 0.5)) / (1.0 + c3 * m) ...
                * x * (c1 * w + c2 * x * sqrt(w))) + xm;

            d2t2 = d2t / 2.0;

            if (x >= 1.0)
                n = 1;

                [dt, d2t, d3t, t0] = tlamb(m, q, qsqfm1, 0, 0);


                tdiff0 = t0 - tmin;

                tdiff = tin - t0;

                if (tdiff <= 0.0)
                    x = xm - sqrt(tdiffm / (d2t2 - tdiffm ...
                        * (d2t2 / tdiff0 - 1.0 / xm^2)));
                else
                    x = -tdiff / (tdiff + 4.0);

                    w = x + c0 * sqrt(2.0 * (1.0 - thr2));

                    if (w < 0.0)
                        x = x - sqrt(d8rt(-w)) * (x + sqrt(tdiff / (tdiff +
1.5 * t0)));
                    end

                    w = 4.0 / (4.0 + tdiff);

                    x = x * (1.0 + (1.0 + m + c42 * (thr2 - 0.5)) ...
                        / (1.0 + c3 * m) * x * (c1 * w - c2 * x * sqrt(w)));

                    if (x <= -1.0)
                        n = n - 1;
```

```
                        if (n == 1)
                            x = xpl;
                        end
                    end
                end
            end
        end
    else
        n = -1;

        termflag = true;
    end
end

% now have a starter, so proceed by halley

if (termflag ~= true)
    for i = 1:1:3
        [dt, d2t, d3t, t] = tlamb(m, q, qsqfm1, x, 2);

        t = tin - t;

        if (dt ~= 0.0)
            x = x + t * dt / (dt * dt + t * d2t / 2.0);
        end
    end

    if (n ~= 3)
        % exit if only one solution, normally when m = 0
        return
    end

    n = 2;

    xpl = x;

    % second multi-rev starter

    [dt, d2t, d3t, t0] = tlamb(m, q, qsqfm1, 0, 0);

    tdiff0 = t0 - tmin;

    tdiff = tin - t0;

    if (tdiff <= 0.0)
        x = xm - sqrt(tdiffm / (d2t2 - tdiffm * (d2t2 / tdiff0 - 1.0 /
 xm^2)));
    else
        x = -tdiff / (tdiff + 4.0);

        w = x + c0 * sqrt(2.0 * (1.0 - thr2));

        if (w < 0.0)
```

```
        x = x - sqrt(d8rt(-w)) * (x + sqrt(tdiff / (tdiff + 1.5 * t0)));
    end

    w = 4.0 / (4.0 + tdiff);

    x = x * (1.0 + (1.0 + m + c42 * (thr2 - 0.5)) / (1.0 + c3 * m) ...
        * x *(c1 * w - c2 * x * sqrt(w)));

    if (x <= -1.0)
        n = n - 1;

        if (n == 1)
            % no finite solution for x < xm
            x = xpl;
        end
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%

function d8rt = d8rt(x)

% Gooding lambert support function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d8rt = sqrt(sqrt(sqrt(x)));
```

Published with MATLAB® R2021b