A Lambert's Problem solver

Mont Blanc - montblanc2012@icloud.com

July 7, 2022

1 *Introduction*

In celestial mechanics, Lambert's problem is concerned with the determination of an orbit from two position vectors - an initial point, $\mathbf{R}_1$; and a final point, $\mathbf{R}_2$ with radius r_1 and r_2, respectively; a known time-of-flight, ΔT; and a central Keplerian gravitational field of known gravitational parameter, μ. This problem has important applications in the areas of rendezvous, targeting, guidance, and preliminary orbit determination.

Fast and robust packages to solver packages exist. There is, arguably, little need for another one. However, Lambert's Problem is an interesting mathematical problem and a detailed study of it is, of itself, rewarding. Moreover, it is sometimes convenient to have 'on hand' a robust and accurate solver that can be quickly represented in a variety of computer languages[1] rather than use a 'black box' solution from an external code repository.

[1] In this note, we present a demonstration Python implementation of the algorithm

Consequently, the purpose of this note is to set out a solution algorithm that solves the multi-period Lambert's Problem. The algorithm presented (and illustrated) herein is based upon the Kustaanheimo-Stiefel (KS) regularising transformation[2]. Proof of the maths underpinning the algorithm is not given. However, if any reader is sufficiently interested in understanding the proof, they should contact the author at the email address given above.

[2] See, for example, https://arxiv.org/abs/0803.4441 - "Interpreting the Kustaanheimo-Stiefel transform in gravitational dynamics"

2 *The algorithm inputs*

The algorithm has the standard inputs to Lambert's Problem. These are[3]:

[3] In this note, we consider anti-clockwise (prograde) orbits only. A trivial extension to the algorithm permits calculation of the clockwise (retrograde) orbits.

- $\mathbf{R}_1 = (R_{1,x}, R_{1,y}, R_{1,z})$ - the initial position vector;
- $\mathbf{R}_2 = (R_{2,x}, R_{2,y}, R_{2,z})$ - the final position vector;
- ΔT - the transfer time from $\mathbf{R}_1$ to $\mathbf{R}_2$;
- μ - the gravitational parameter for the central gravitating body; and
- w - the winding number of the solution orbit.

3 The algorithm

To implement the algorithm calculate the following:

$$
\begin{aligned}
r_1 &= \|\mathbf{R}_1\| \\
r_2 &= \|\mathbf{R}_2\| \\
A &= r_1 + r_2 \\
B &= (-1)^w \sqrt{2\,(\mathbf{R}_1 \cdot \mathbf{R}_2 + r_1\,r_2)} \\
\delta t &= \Delta T \sqrt{2\,\mu / A^3}
\end{aligned}
$$

If $(\mathbf{R}_1 \times \mathbf{R}_2) \cdot (0, 0, 1) < 0$ then set B to $-B$[4].

[4] This selects the anti-clockwise (prograde) orbits. To select the retrograde orbits, we would apply the rule: if $(\mathbf{R}_1 \times \mathbf{R}_2) \cdot (0, 0, 1) > 0$ then set B to $-B$.

Define a function $g(Y)$ such that:

$$
\begin{aligned}
f &= \sin(Y)/Y \\
X &= \cos(Y) \\
\phi &= B/A \\
a &= 1 - \phi\,X \\
b &= \phi - X
\end{aligned}
$$

$$g(Y) = \delta t\,|f|\,(1 - X^2) - \sqrt{a}\,(a + b\,f)$$

The solutions of Lambert's Problem are given as the roots of the transcendental function g[5]:

[5] Multiple methods exist for finding roots of transcendental functions. In this note, we use a root-finding algorithm based on approximating the function g with Chebyshev polynomials and then finding the (real) eigenvalues of the associated comrade matrix.

$\Delta T > T_p$	$g(Y^*) = 0$	$w\,\pi < Y^* < (w+1)\,\pi$
$\Delta T < T_p$	$g(i\,Y^*) = 0$	$0 < Y < \cosh^{-1}(A/B) \quad B > 0$
$\Delta T < T_p$	$g(i\,Y^*) = 0$	$0 < Y < \infty \quad B \leq 0$

where:

$$T_p = \frac{1}{3}\sqrt{\frac{A - B}{2\,\mu}}\,(2\,A + B)$$

For parabolic orbits[6], where $\Delta T = T_p$, $Y^* = 0$. There are many ways of finding these roots. Some methods are robust but not fast; others are fast but not robust. We do not specify any particular method for finding the roots of this equation here.

[6] The usual trichotomy of elliptical, parabolic and hyperbolic orbits applies. If $\Delta T > T_p$, solutions to Lambert's problem are elliptical; if $\Delta T = T_p$ solutions are parabolic; and if $\Delta T < T_p$, solutions are hyperbolic. For parabolic and hyperbolic orbits, the winding number w must be 0.

GIVEN A ROOT Y^* OF THE FUNCTION g as defined above, we calculate the quantity X^* as[7]:

$$X^* = \cos(Y^*) \qquad \Delta T > T_p$$
$$X^* = \cosh(Y^*) \qquad \Delta T < T_p$$

[7] This purpose of defining this function, and for finding the values X^* related to the roots of this expression is a product of imposing the time-of-flight constraint.
However, there are variants of Lambert's problem that replace the time-of-flight constraint with, say, a constraint on the periapsis radius, r_p instead. In this particular case, for example, the values of X^* can be computed analytically as:

$$X^* = \frac{B \pm 2\sqrt{(r_1 - r_p)\,(r_2 - r_p)}}{2\,r_p}$$

These cases arise when it matters less 'when' we arrive at $\mathbf{R_2}$ and more 'how' we arrive. For example, let's suppose that we we want our destination $\mathbf{R}_2$ to be at orbital periapsis. Then, we must have $r_2 = r_p$. And from the above expression, we require that $X^* = B/(2\,r_p)$. We then proceed with calculating the initial and final velocity vectors as per the main text (and setting the winding number w to zero).

The initial and final position vectors corresponding to the root, Y^*, are calculated as follows:

$$f_1 = \sqrt{(R_{1,z} + r_1)/2}$$
$$f_2 = \sqrt{(R_{2,z} + r_2)/2}$$

$$\chi_1 = R_{1,x}/f_1/2$$
$$\chi_2 = R_{2,x}/f_2/2$$
$$\zeta_1 = R_{1,y}/f_1/2$$
$$\zeta_2 = R_{2,y}/f_2/2$$

$$c = 2\,(+\chi_1\,\chi_2 + \zeta_1\,\zeta_2 + f_1\,f_2)/B$$
$$s = 2\,(-\chi_1\,\zeta_2 + \chi_2\,\zeta_1)/B$$

And calculate the following quaternions:

$$\mathcal{K} = [0, 0, 0, 1]$$
$$\mathcal{Q}_1 = [0, \chi_1, \zeta_1, f_1]$$
$$\mathcal{Q}_2 = [s\,f_2, c\,\chi_2 - s\,\zeta_2, c\,\zeta_2 + s\,\chi_2, c\,f_2]$$

$$\mathcal{V}_1 = (-1)^w \sqrt{\frac{2\,\mu}{A - B\,X^*}}\;(\mathcal{Q}_2 - X^*\,\mathcal{Q}_1)\;\mathcal{K}\,\mathcal{Q}_1^\dagger\;/\;r_1$$
$$\mathcal{V}_2 = (-1)^w \sqrt{\frac{2\,\mu}{A - B\,X^*}}\;(X^*\,\mathcal{Q}_2 - \mathcal{Q}_1)\;\mathcal{K}\,\mathcal{Q}_2^\dagger\;/\;r_2$$

The initial and final velocity vectors are encoded in the quaternions $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that:

$$\mathcal{V}_1 = [0, v_{1,x}, v_{1,y}, v_{1,z}]$$
$$\mathcal{V}_2 = [0, v_{2,x}, v_{2,y}, v_{2,z}]$$

from which we can write:

$$\mathbf{V}_1 = (v_{1,x}, v_{1,y}, v_{1,z})$$
$$\mathbf{V}_2 = (v_{2,x}, v_{2,y}, v_{2,z})$$

This completes the solution of the Lambert's Problem.

4 Some Python code

IN THIS SECTION, and as an illustration of the above algorithm, we present some Python code[8] for solving the multi-revolution Lambert's Problem.

[8] In this note, Python code is based upon Python 3.

The first step is to import the following packages:

```
import numpy as np
import quaternion
import math
```

Next, we calculate the values of Chebyshev polynomials at the Chebyshev interpolation points. The resulting array is used in finding the roots of the function, $g(Y)$. In this particular case, Chebyshev functions up to order 50 are calculated. These tables can be re-used in repeated application of the Lambert solver and do not need to be calculated upon each call of the routine.

```
n       = 50
m       = n + 1
d       = 2.0 / m
l       = np.arange(m)
pts     = np.cos((l + 0.5)/m * np.pi)
T       = np.empty([m, m])
T[0]    = np.ones(m)
T[1]    = pts
for idx in range(2, m):
    T[idx] = 2 * pts * T[idx-1] - T[idx-2]
```

We then define the core Lambert's Problem solver as:

```
def lambert(R1, R2, delT, mu, w):

    z          = 1 - 2 * (w % 2)
    r1         = np.linalg.norm(R1)
    r2         = np.linalg.norm(R2)
    A          = r1 + r2
```

```
alpha       = np.dot(np.cross(R1, R2),[0.0, 0.0, 1.0])
B           = +z*np.sqrt(2.0 * (np.dot(R1, R2) + r1 * r2))
if alpha < 0:
    B       = -B

phi         = B / A
dt          = delT * math.sqrt(2.0 * mu / A) / A
Y           = np.pi * (w + 0.5 * (pts + 1.0))
X           = np.cos(Y)
a           = 1.0 - phi * X
b           = phi - X
f           = np.sin(Y) / Y
gpts        = dt * abs(f) * (1 - X*X) - np.sqrt(a) * (a + f*b)
cheby       = np.dot(T, gpts)
cheby      *= d
cheby[0]    = cheby[0] * 0.5

# calculate the roots of the function 'g'
eigs        = np.polynomial.chebyshev.chebroots(cheby)
eigs        = eigs[np.isreal(eigs)].real
eigs        = eigs[list(map(lambda x: -1 < x and x < 1, eigs))]
yroots      = np.pi * (w + 0.5 * (eigs + 1))
if w == 0:
    yroots = [np.amax(yroots)]
X           = np.cos(yroots)

# convert the roots to velocity vectors
f1          = np.sqrt((R1[2] + r1) / 2.0)
f2          = np.sqrt((R2[2] + r2) / 2.0)
xi1         = R1[0] / f1 / 2.0
xi2         = R2[0] / f2 / 2.0
ze1         = R1[1] / f1 / 2.0
ze2         = R2[1] / f2 / 2.0
c           = 2 * ( +xi1 * xi2 + ze1 * ze2 + f1 * f2) / B
s           = 2 * ( -xi1 * ze2 + xi2 * ze1) / B
K           = np.quaternion(    0,              0,              0,    1)
Q1          = np.quaternion(    0,            xi1,            ze1,   f1)
Q2          = np.quaternion( s*f2, c*xi2 - s*ze2, c*ze2 + s*xi2, c*f2)
V1          = list(map(lambda x: z*np.sqrt(2 * mu / (A - B*x)) * _
    (Q2 - x*Q1) * K * np.conj(Q1) / r1, X))
V2          = list(map(lambda x: z*np.sqrt(2 * mu / (A - B*x)) * _
    (x*Q2 - Q1) * K * np.conj(Q2) / r2, X))

return [V1, V2]
```

Calls to this function return the initial and final velocity vectors as lists of quaternions.

5 Some worked examples

IN THIS SECTION, we will consider two worked examples of application of the algorithm. The first worked example is a standard time-of-flight constrained Lambert's Problem as per the Python code described above.

The second worked example, replaces the time-of-flight constraint with the requirement that the destination position vector $\mathbf{R}_2$ be located at orbital periapsis. As per the sidenotes above, we replace the calculation of the $X*$ from the roots of the function $g(Y)$ with the straightforward analytical expression, $X^* = B/(2\,r_2)$ ans set the winding number, w to 0.

A standard time-of-flight constrained Lambert's Problem

FOR SIMPLICITY, let's suppose that we have a Lambert's problem with:

$$\begin{aligned}
\mathbf{R}_1 &= (1,\, 0,\, 0) \\
\mathbf{R}_2 &= (0,\, 1,\, 0) \\
\Delta T &= 9\pi/2 \\
\mu &= 1 \\
w &= 2
\end{aligned}$$

With these parameters, then we call the Python Lambert solver routine as:

```
sol = lambert([1,0,0], [0,1,0], 9*np.pi/2, 1, 2)
```

We extract the solution initial velocity vectors as:

```
list(map(lambda k: [k.x, k.y, k.z], sol[0]))
```

which returns:

```
[[-7.494005416219835e-15, 1.0000000000000038, 1.6653345369377348e-16],
 [0.5624724952837518, 0.7575581627145153, 2.220446049250313e-16]]
```

After a little truncating and rounding of the results, the two solution initial velocity solutions are:

$$\mathbf{V}_{1,a} = [+0.0000000, +1.0000000, 0.0000000]$$
$$\mathbf{V}_{1,b} = [+0.5624725, +0.7575582, 0.0000000]$$

Similarly, we can can extract the solution final velocity vectors as:

```
list(map(lambda k: [k.x, k.y, k.z], sol[1]))
```

which returns:

```
[[-1.0000000000000038, 7.382983113757291e-15, -3.3306690738754696e-16],
 [-0.7575581627145153, -0.5624724952837518, -1.8735013540549517e-16]]
```

Again, after a little truncating and rounding of the results, the two solution final velocity solutions are:

$$\mathbf{V}_{2,a} = [-1.0000000, +0.0000000, 0.0000000]$$
$$\mathbf{V}_{2,b} = [-0.7575582, -0.5624725, 0.0000000]$$

To illustrate the solution trajectories, we plot them below:

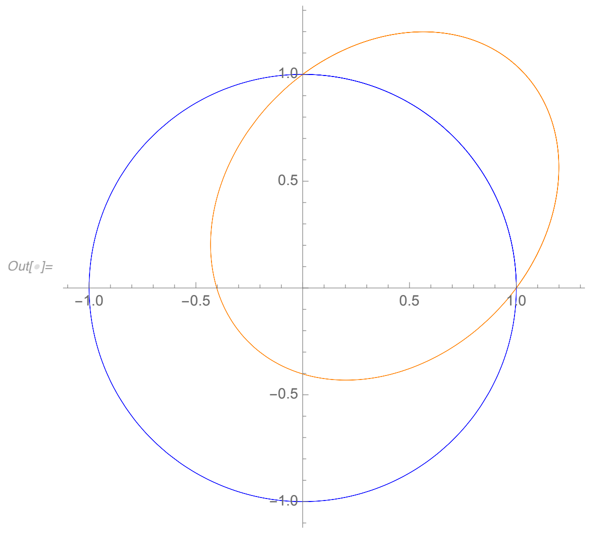

Figure 1: The blue trajectory is the first solution to the multi-period Lambert's Problem; and the orange trajectory is the second. In accordance with the input requirements, both solution trajectories complete more than two complete orbits ($w = 2$) of the gravitating body before arriving at $\mathbf{R}_2$

It is also of some interest to plot the transcendental function g so as to develop an intuitive understanding of its structure. As shown in the plot below, $g(Y)$ is bounded continuous and differentiable almost everywhere. In particular, the function is not differentiable for integer multiples of π. This provides a natural division of the function domain into a serious of consecutive intervals $(k\,\pi,\,(k+1)\,\pi)$. On each of these intervals, the function g is bounded, continuous and differntiable. This makes it a good candidate for Chebyshev approximation on each of these intervals. This provides the conceptual foundation for the algorithm used in the Python code above.

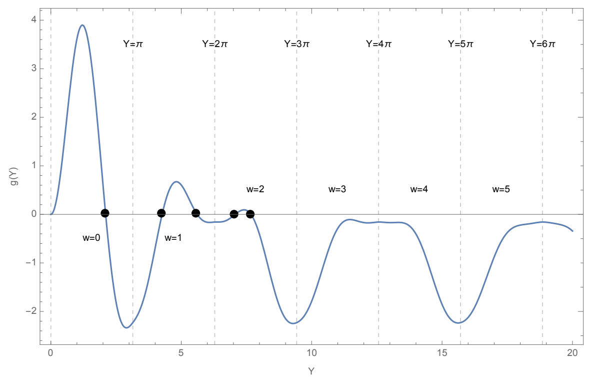

Figure 2: A plot of the function $g(Y)$ for the example given in the main text. The five of the function are identified with dots. For $w = 0$ (the interval $0 < Y < \pi$) there is one non-zero root; for $w = 1$ (the interval $\pi < Y < 2\,\pi$) there are two roots; and for $w = 2$ (the interval $2\,\pi < Y < 3\,\pi$) there are again two roots. For higher winding numbers there are no further roots.

On the interval $(0,\,\pi)$, there is one root. Although there is a root at $Y = 0$, this value is not included in the solution set. Consequently, when the winding number is 0, there is just one valid prograde solution. In each of the intervals $(\pi,\,2\,\pi)$ and $(2\,\pi,\,3\,\pi)$ there are a further two roots - each of which corresponds to a valid prograde solution of the example problem. For higher winding numbers there are no further roots and, hence, no further solutions.

A modified Lambert's Problem - arriving at orbital periapsis at $\mathbf{R}_2$

LET'S CONSIDER LAMBERT'S PROBLEM with the following initial and final velocity vectors.

$$\begin{aligned}\mathbf{R}_1 &= (10,\,0,\,0)\\ \mathbf{R}_2 &= (0,\,1,\,0)\\ \mu &= 1\end{aligned}$$

This emulates a situation in which one is targeting arriving at orbital periapsis (presumably for orbit insertion) while still at some distance from the target body. Rather than use the g function to calculate X^* by solving for the roots of g, we can by pass the time-consuming root-finding algorithm by modifying the Python code so that it calculates X^* directly from the expression:

$$X^* = \frac{B}{2\,r_p}$$

The relevant code snippet for the revised algorithm is the straightforward:

```
def lambert2rp(R1, R2, mu):

    r1          = np.linalg.norm(R1)
    r2          = np.linalg.norm(R2)
    A           = r1 + r2
    B           = np.sqrt(2.0 * (np.dot(R1, R2) + r1 * r2))
    alpha       = np.dot(np.cross(R1, R2),[0.0, 0.0, 1.0])
    if alpha < 0: B = -B

    # calculate X
    X           = np.array([B/2/r2])

    # calculate the 'initial' and 'final' velocity vectors
    f1          = np.sqrt((R1[2] + r1) / 2.0)
    f2          = np.sqrt((R2[2] + r2) / 2.0)
    xi1         = R1[0] / f1 / 2.0
    xi2         = R2[0] / f2 / 2.0
    ze1         = R1[1] / f1 / 2.0
    ze2         = R2[1] / f2 / 2.0
    c           = 2 * ( +xi1 * xi2 + ze1 * ze2 + f1 * f2) / B
    s           = 2 * ( -xi1 * ze2 + xi2 * ze1) / B
    K           = np.quaternion(    0,              0,              0,    1)
    Q1          = np.quaternion(    0,            xi1,            ze1,   f1)
    Q2          = np.quaternion( s*f2, c*xi2 - s*ze2, c*ze2 + s*xi2, c*f2)
    V1          = list(map(lambda x: np.sqrt(2 * mu / (A - B*x)) * _
        (Q2 - x*Q1) * K * np.conj(Q1) / r1, X))
    V2          = list(map(lambda x: np.sqrt(2 * mu / (A - B*x)) * _
        (x*Q2 - Q1) * K * np.conj(Q2) / r2, X))

    return [V1, V2]
```

Note here that no root-finding algorithm is employed. Running this Python code yields the following initial and final velocity vectors of the solution[9]:

[9] There is also a retrograde solution - however, this solution is simply the time-reversal of the prograde solution.

$$\mathbf{V}_1 = [-2.8460499, +0.3162278, 0.0000000]$$
$$\mathbf{V}_2 = [-3.1622777, +0.0000000, 0.0000000]$$

Since the final velocity vector, $\mathbf{V}_2$ is orthogonal to the final position vector, $\mathbf{R}_2$, we know that the solution is indeed at orbital periapsis at the final position vector - as required.

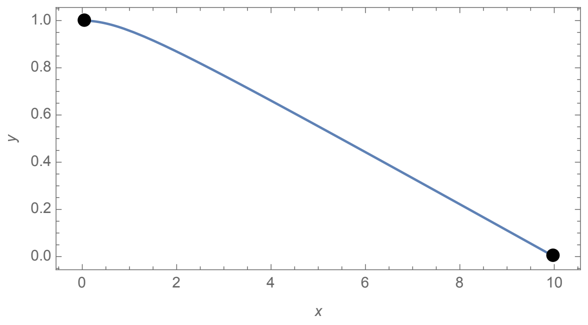

Figure 3: A plot of the prograde trajectory from $\mathbf{R}_1 = (10, 0, 0)$ to $\mathbf{R}_2 = (0, 1, 0)$ calculated using direct integration of the initial velocity vector $\mathbf{V}_1 = [-2.8460499, +0.3162278, 0.0000000]$. Visually, this confirms the arrival at $\mathbf{R}_2$ at orbital periapsis.

In this particular case, we note that $X > 1$ which means that the transfer orbit is hyperbolic[10]. If we had found, on the other hand, that $X < -1$[11] the solution would also have been hyperbolic but no valid transfer from $\mathbf{R}_1$ and ends at orbital periapsis at $\mathbf{R}_2$ exists. Consequently, if the trajectory is hyperbolic, and for there to be a valid transfer orbit, we require $B > 0$.

[10] An elliptical orbit solution would have $-1 < X < 1$.

[11] Also a hyperbolic transfer orbit.

Moreover, for this example, we can calculate the time-of-flight as[12]

[12] This formula applies to hyperbolic orbits. We know that the solution is hyperbolic because $X > 1$. If the solution to the problem had been elliptical, the time-of-flight would have given by the expression:

$$\Delta T = 2\sqrt{\frac{a^3}{\mu}}\left(\cos^{-1} X_1 + X_2\sqrt{1 - X_1^2}\right)$$

$$\begin{aligned}
X_1 &= X \\
X_2 &= \frac{B - A\,X_1}{A - B\,X_1} \\
a &= \frac{A - B\,X_1}{2\,(1 - X_1^2)} \\
\Delta T &= -2\sqrt{\frac{-a^3}{\mu}}\left(\cosh^{-1} X_1 + X_2\sqrt{X_1^2 - 1}\right)
\end{aligned}$$

For this example, we calculate the time-of-flight as 3.429961.

6 Conclusion

In this note, we have described an algorithm to solve the standard Lambert's Problem (and variants thereof). This algorithm is based on the KS regularisation transformation. Proofs are not given in this note but are available upon request.

The standard time-of-flight representation of Lambert's Problem inevitably requires finding the roots of a transcendental function. In the Python representation of the elliptical multi-revolution algorithm set out in this note, we need to find the real roots of a function $g(Y)$. As a choice, roots are found by approximating the function g on a prescribed interval using Chebyshev interpolation and then solving for the roots by calculating the eigenvalues of the associated comrade matrix of the resulting Chebyshev polynomial approximation. This approach is robust but is not cutting-edge in terms of speed. Having said that, the algorithm certainly isn't sluggish and, given sufficient effort in optimising coding of the calculation of eigenvalues of a very sparse comrade matrix, the overall speed of the algorithm should at least be comparable to other root-finding approaches.

Quaternion arithmetic has been employed to calculate the initial and final velocity vectors of the transfer trajectory. This avoids using rotation matrices to convert the raw roots of the transcendental function g to solution values of Lambert's Problem.

The same algorithm can be easily modified to solve variants of the standard time-of-flight constrained Lambert's Problem. In this note, an example is given of a modified version of Lambert's Problem where the time-of-flight constraint is replaced by the requirement that the transfer orbit achieve orbital periapsis upon arrival at $\mathbf{R}_2$. Solution of this modified problem does not require a time-consuming identification of the roots of a transcendental function. Other non-time-of-flight constrained Lambert's Problem can be readily constructed.