Table of Contents


```
function [v1,v2,dTheta] = LambertSolver( r1, r2, TOF, mu, dir )

% Solve Lambert's problem.
%
%  Given two position vectors, r1 and r2, and the time of flight
 between
%  them, TOF, find the corresponding Keplerian orbit.
```

Input checking

```
% direction
if( nargin<5 )
  dir = 'pro';
end

% Check validity of "dir" input
if( ~ischar(dir) )
  error('Input for direction "dir" must be a string, either ''pro'' or
 ''retro''.');
end

% mu
if( nargin<4 )
  mu = 398600.44;
  warning('Using Earth gravitational constant, mu = %f.',mu);
end

% TOF
if( TOF<=0 )
  error('The time of flight TOF must be >0.')
end
```

Calculate the position vector magnitudes

```
r1m = sqrt(r1'*r1);
r2m = sqrt(r2'*r2);
```

Calculate delta-theta

```
r1CrossR2 = cross(r1,r2);
arg = r1'*r2/r1m/r2m;
if( arg>1 )
  angle = 2*pi;
elseif( arg<-1 )
  angle = pi;
else
  angle = acos( arg );
end
switch lower(dir)
  case {'pro','prograde'}
    if( r1CrossR2(3) >= 0 )
      dTheta = angle;
    else
      dTheta = 2*pi-angle;
    end
  case {'retro','retrograde'}
    if( r1CrossR2(3) < 0 )
      dTheta = angle;
    else
      dTheta = 2*pi-angle;
    end
  otherwise
    error('Unrecognized direction. Use either ''pro'' or ''retro''.')
end

if( abs( abs(dTheta)-2*pi ) < 1e-8 )
  % this is a rectilinear orbit... not supported.
  v1 = nan;
  v2 = nan;
  return;
end
```

Calculate "A"

```
A = sin(dTheta)*sqrt(r1m*r2m/(1-cos(dTheta)));

% Inline function handles, all functions of "z"
yf  = @(z) r1m+r2m+A*( z.*stumpS(z)-1 )./sqrt(stumpC(z));
Ff  = @(z) ( yf(z)./stumpC(z) ).^1.5 .* stumpS(z) + A*sqrt(yf(z)) -
 sqrt(mu)*TOF;
dFf = @(z) LambertFPrimeOfZ( r1m, r2m, dTheta, z );
```

Solve for z ...

```
% Initial guess for z...
z0s = FindInitialZGuessForLambert(yf,Ff);

% consider each approx. crossing and solve for exact z value at
 each...
```

```
% terminate as soon as we have a good answer.
success = 0;
for i=1:length(z0s)
  [zs,success] = NewtonRhapsonSolver( Ff, dFf, z0s(i), 1e-6 );
  if( success )
    break;
  end
end
if( ~success )
  warning('Newton method did not find a solution in LambertSolver.')
  v1=nan;
  v2=nan;
  return;
end
```

Calculate "y"

```
%y = yf(zs);
```

Calculate the Lagrange Coefficients

```
[f,g,~,gdot] = LagrangeCoeffZ(r1m,r2m,dTheta,zs,mu);
```

Calculate v1 and v2

```
v1 = 1/g*(r2-f*r1);
v2 = 1/g*(gdot*r2-r1);
```

Check result...

```
% Calculate the COE from [r1,v1]
[a1,i1,W1,w1,e1,th1] = OrbitalElementsFromRV(r1,v1,mu);

% Calculate the COE from [r2,v2]
[a2,i2,W2,w2,e2,th2] = OrbitalElementsFromRV(r2,v2,mu);

if( norm([1,cos([i1,W1,w1]),e1]-[a2/a1,cos([i2,W2,w2]),e2])>1e-8 )
  warning('Orbital elements from [r1,v1] do not match with those from
 [r2,v2].')
  disp([a1,i1,W1,w1,e1,th1; a2,i2,W2,w2,e2,th2]')
end
```

Published with MATLAB® R2019b