HW 1

Topocentric Coordinates, Angles-Only IOD, Minimum Energy and Lambert's Solution

Justin Self

AERO557: Advanced Orbital Mechanics

February 1, 2024

Problem 1

Topocentric Coordinates.

HST TLEs:

```
1 20580U 90037B 24010.28323428 .00005912 00000-0 29430-3 0 9991
2 20580 28.4707 327.8603 0002613 104.9653 10.4281 15.15476605652889
```

Predict the look angles (RA/DEC and Az/El) and associated times for SLO (coordinates listed below) on Jan 11.
Here are the look angles from Heaven's Above:

```
Date Mag time. El Az. Time. El. Az. Time. El Az. Pass type
11 Jan 2.0 18:50:29 10° SW 18:53:58 28° S 18:54:32 27° SSE visible
```

Discuss the differences.

SLO Coordinates: 35.3540 north 120.3757 west 105.8 m altitude

Solution.

See Appendix A for results and code for Problem 1

Given initial Two Line Elements (TLEs) for the Hubble Space Telescope (HST), look angles were calculated using a blend of homemade, AERO557 provided, publicly published, and native `MATLAB` code. Since the problem statement asked for look angles and times for *January 11*, the simulation was modelled to consider visible HST passes between 11 Jan 2024 00:00:00 (local time) to 11 Jan 2024 23:59:59 (local time). Consequently, the HST orbit was propagated from the epoch date (10 January 2024 06:47:51, UTC) to the mission start date (11 Jan 08:00:00, UTC).

The initial orbit propagation method was a two-body motion propagator using the `MATLAB ODE45` ODE solver. Orbital perturbations were *not* simulated as a condition of the problem given, which very likely led to errors in look angle predictions, specifically with respect to time and elevation. Fig. 1 shows the osculating orbit propagated from the TLE retrieval epoch to mission start time and positions of the HST at each time.

Once the orbit was propagated to the mission start time (00:00:00 Jan 11, site time) using the two-body propagator, the simulation then used a universal variable propagator at one-second time intervals. Fig. 2 shows a graphical representation of conditions needed for viewing the HST on pass visible from SLO. THe conditions are:

- The spacecraft must be illuminated by the sun
- The site must be in darkness
- Spacecraft must be above the horizon ($180^\circ \leq el \leq 0^\circ$)

The first two conditions are plotted against mission time in Fig. 2, with two candidate time blocks highlighted in black.

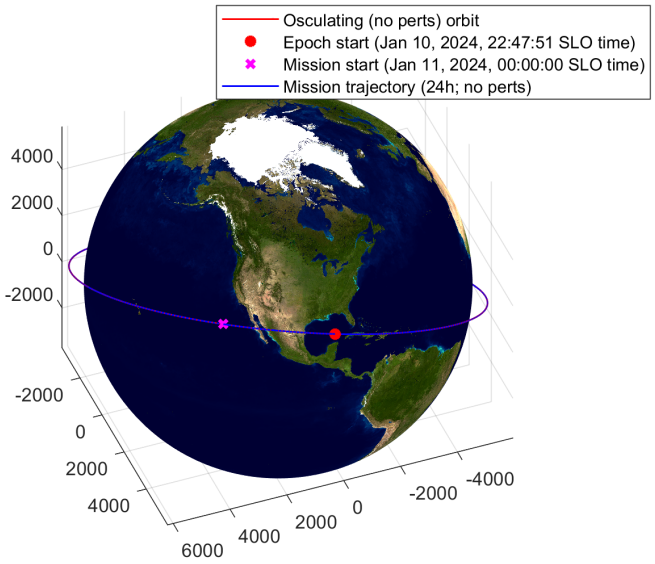

Figure 1: Osculating orbit trajectory (ECI) of HST starting at epoch (TLE retrieval) and mission start time (00:00:00 11 Jan, 2024; Pacific Standard Time). HST was propagated until 11 Jan 23:59:59 without perturbations, which led to small errors in elevation, azimuth, and viewing times. Planet3D visualization code courtesy of Tamas Kis, throughout.

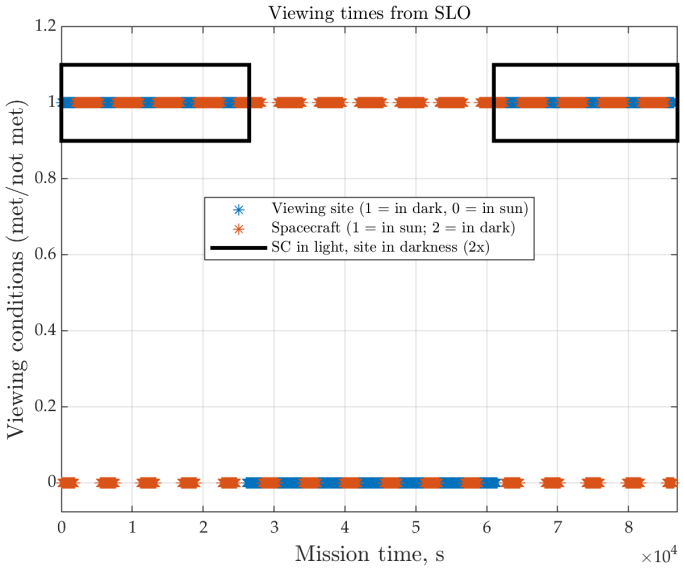

Figure 2: Graphic representation of viewing conditions from SLO, California. Zero means the spacecraft is either in eclipse or the site is in daylight. One means the spacecraft is in the sun or the site is in darkness. No masking is applied, so any viewing elevation > 0 will be captured. The black boxes highlight HST viewing candidate regions.

After propagating the orbit and computing changing conditions at each time step, the data was filtered the third condition (spacecraft must be between 0° and 180°). Fig. 3 shows the elevation and associated times of HST flybys from the local SLO frame. Note that there

are three candidate flybys on 11 Jan 2024:

a) 17:11 - 17:21 max el: 10.3°

b) 18:50 - 18:57, max el: 23.8°

c) 20:31 - 20:32, (no max elevation reached)

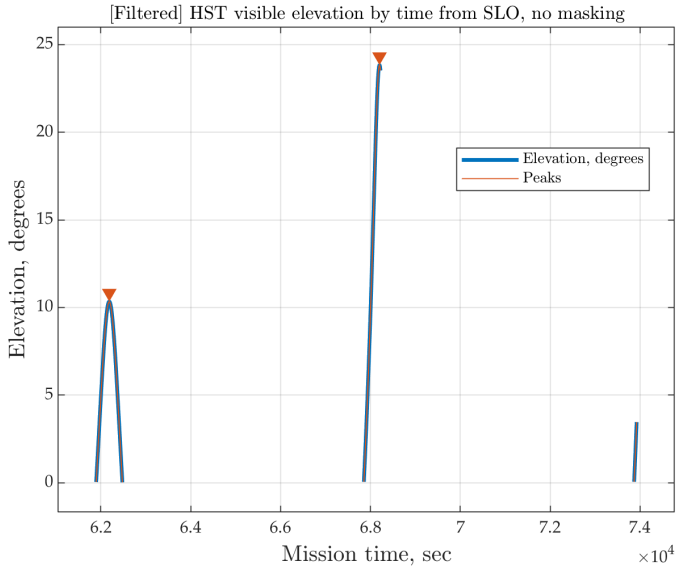

Figure 3: Elevation peaks and associated times for HST viewing from SLO on 11 Jan, 2024. This plot comes from the right-hand side block highlighted in Fig. 3. This means the only visible passes of HST from the site between 11 Jan (00:00:00 local) and 12 Jan (23:59:59 local) occurred at these times.

Of the three candidates, the second view corresponds closely with the data given by Heavens-Above (see Fig. 4). Notably, Heavens-Above applies a 10° mask to their view angle calculations, and ours does not. Thus, View 3 would not show up on H-A datasets. It is likely that this is why View 1 (max elevation ~ 10°) doesn't show up on H-A.

The reason our viewing times are off could be due to several reasons. First, orbital perturbations were not considered in this simulation. Over a time span of 24 hours, the COEs would have changed due to zonal harmonics, solar radiation pressure, n-body effects, and atmospheric drag. Secondly, HST could also have performed a thrust maneuver during the time span after initial TLE retrieval. Thirdly, H-A doesn't start recording a pass until the sun is 6° below the horizon (both rising and setting)[1]:

```
We have a cut-off of -6° sun altitude for our predictions, so that
satellite passes are only predicted when the sky is reasonably dark.
However, these bright satellites can sometimes be seen when the
sun is higher, but this explains why they aren't in the list.
```

Fourthly, our model does not apply the same 10° masking that H-A does. This would skew the viewing times. These are all plausible reasons why our simulation does not match

exactly with the H-A predictions. Figs. 5, 6, 7, and 8 show our simulation results, which agree reasonably with H-A predictions given in Fig. 4.

Date	Mag	time.	El	Az.	Time.	El.	Az.	Time.	El	Az.	Pass type
11 Jan	2.0	18:50:29	10°	SW	18:53:58	28°	S	18:54:32	27°	SSE	visible

Figure 4: Given viewing times/angles for HST from Heavens-Above. Note that the Heavens-Above HST pass algorithm appears to build in a 10° elevation mask, so published passes begin at 10°.

	View 1	View 2	View 3
Start	11-Jan-2024 17:11:35	11-Jan-2024 18:50:53	11-Jan-2024 20:31:08
Peak	11-Jan-2024 17:16:26	11-Jan-2024 18:56:41	11-Jan-2024 20:32:04
Stop	11-Jan-2024 17:21:18	11-Jan-2024 18:57:02	11-Jan-2024 20:32:04

Figure 5: Of the three visible passes, View 2 matches Heavens-Above data shown in Fig. 4. This is likely due to the 10° elevation mask, which would remove View 1 and View 3. The times are similar, but not exact. This is likely due to perturbational effects, not considered in this model.

	View 1	View 2	View 3
Start az [deg]	208.661795516048	236.701274199645	253.922750831431
Peak az [deg]	157.092014219895	168.413783014575	250.947287773236
Stop az [deg]	105.596469629532	159.966604786374	250.947287773236

Figure 6: Azimuths for three calculated HST viewing times from SLO between 11 Jan and 12 Jan, 2024.

	View 1	View 2	View 3
Start az direction	{'SSW'}	{'SW' }	{'WSW'}
Peak az direction	{'SSE'}	{'S' }	{'WSW'}
Stop az direction	{'ESE'}	{'SSE'}	{'WSW'}

Figure 7: Compass directions for three calculated HST viewing times from SLO between 11 Jan and 12 Jan, 2024. Note that View 2 agrees with the H-A data given in Fig. 4.

	View 1	View 2	View 3
Start el [deg]	0.0359222620314736	0.0456622012614463	0.0480099784863766
Peak el [deg]	10.3357511324859	23.8335804788815	3.45902152009376
Stop el [deg]	0.0178591802397283	23.5284460587052	3.45902152009376

Figure 8: Predicted elevation for visible HST passes as seen from SLO from 11 Jan to 12 Jan, 2024. Note that our predictions do not include elevation masking, as H-A does (see Fig. 4). Max elevations are also slightly off, likely due to perturbational effects (not considered in this model).

Problem 2

Gauss, Extended Gauss, and Double-R methods for angles-only initial orbit determination.

Calculate the position and velocity vectors from the double-r method and Gauss' method (non-extended and extended) for topocentric angles only observations.

In addition, compare the resulting COEs from the state vectors as well. Discuss the differences and thoughts on each of the methods. Please note, in the future, the Gauss method is always run with the extension. Finally, pick what TLE object might be close for this object (assume no perturbations) and just compare the COES.

```
Observation RA (degs) DEC (degs) Date (UT) Time (UT)
1 30.381 23.525 03-25-2013 03:10:30.032
2 65.134 0.774 03-25-2013 03:15:20.612
3 99.976 -30.44 03-25-2013 03:20:32.777
```

Observations are taken at Cal Poly Observatory:
35.30 north, 120.66 west, 105.8 m altitude

```
TLE options:
1 25623U 99004C 13109.04882318 -.00000094 00000-0 -16690-3 0 1204
2 25623 051.9974 067.7982 0012092 184.1231 215.4516 11.86494224651645

1 25165U 98008D 13083.14572197 -.00000211 00000-0 -12941-2 0 4434
2 25165 052.0160 303.6990 0005433 319.9573 182.6276 12.12023409691559

1 25946U 99058D 13104.16396495 -.00000071 00000-0 19175-3 0 939
2 25946 051.9981 329.5396 0000986 149.5293 353.4996 12.46940793622716
```

Solution.

See Appendix B for results and code for Problem 2

Position and velocity vectors for the middle of three observations were determined using a home-built function called `gauss.m` allowing users to choose between Gauss and Gauss

Extended methods. These methods were compared with the Double-R angles-only method. Tabulated results are shown in Fig. 9. Fig. 10 gives the associated Common Orbital Elements (COEs) from each method.

Gauss' technique using angles-only data is said to work best when the angular separation between observations is less than 60°, but it is shown to perform remarkably well for data separated by 10° or less [2]. In this analysis, the Gauss method (not extended) produced the least accurate (when compared to the Double-R method) results of the three methods utilized. Both Gauss and Gauss extended models utilized the Gibbs initial orbit determination algorithm before iterative processing (if using extended).

The Double r-iteration method is a combination of numerical and dynamical techniques proposed by Escobal (1965) and is more robust than the Gauss method. This method, according to [2], can handle observations that are days apart. This method still utilizes the familiar t_m (transfer method) variable (+1) for the short way, (−1) for the long way (retrograde) transfer trajectory. The initial guess is important in this method. In our case, this method produced the most accurate results, shown in Figs. 9 - 10.

	Gauss	Gauss Extended	Double-R
r2x [km]	-840.054378084647	-823.006650676095	-823.007283505782
r2y [km]	7070.6739188924	7107.45724373507	7107.45587829911
r2z [km]	3698.23881177458	3698.78651819708	3698.78649786563
v2x [km/s]	-5.28758213855778	-5.29401099273387	-5.29400816978208
v2y [km/s]	1.79781989089623	1.80435321941347	1.80435266589727
v2z [km/s]	-4.64317551818809	-4.65391220082394	-4.65390969069674

Figure 9: Position and velocity vectors obtained from three IOD angles-only methods.

	Gauss	Gauss Extended	Double-R
h [km2/s]	58274.1435320561	58604.7693607567	58604.7304178864
inc [deg]	52.0008016878705	51.9239527750232	51.9239576964323
RAAN [deg]	300.715297695514	300.495990828433	300.495997043193
ecc	0.0618157954365597	0.0697770403678169	0.0697757707427007
omega [deg]	144.505649267153	144.798820240989	144.798765881347
True anomaly [deg]	359.697400920804	359.513483519477	359.513535001091
Semi-major axis [km]	8552.18734293876	8658.61243071461	8658.59938178775
Period [s]	7870.94776097365	8018.32531132595	8018.30718535567

Figure 10: COEs obtained from the state vectors obtained by the three IOD angles-only methods.

After computing state vectors and COEs from each method, the next step is to compare the given TLEs with the calculated ones. Relative errors (absolute value of the differences) were plotted and a heatmap generated to compare the two datasets, using the Double-R method as the calculation of choice (see Fig. 11. At first glance, it appears that C is the winner, since there is less error overall. However, it is important to note that all three *inclinations* (A, B, C) are very close to the calculated values. Thus, it is expected that RAAN will be close also. It is clear from Figs. 11 and 12 that both inclination and RAAN agree

reasonably. The eccentricity calculation, although verified against the three IOD methods, did not agree with any of the TLE data. However, all calculation methods and given TLEs give eccentricities near zero.

Since the calculated orbit is very close to circular, any error in argument of perigee may be ignored since ω is inaccurate for highly circular orbits. Furthermore, a is calculated using apogee and perigee radii, and thus a circular orbit will further confound any semi major axis calculations. Period is nearly the same. Revs per day are very close. Thus, TLE B seems to be the correct TLE.

	A	B	C
Err: inc [deg]	0.0734423035677025	0.0920423035677018	0.0741423035677045
Err: RAAN [deg]	232.697797043193	3.2030029568067	29.0436029568067
Err: ecc	0.0685665707427007	0.0692324707427007	0.0696771707427007
Err: omega [deg]	39.3243341186534	175.158534118653	4.73053411865337
Err: n-bar [rev/day]	7.92379852949584e-05	7.92379852949584e-05	0.00012319597073595 6
Err: Semi-major axis [km]	538.563868872739	538.563868872739	803.148351942793
Err: Period [h]	0.204541699405458	0.204541699405458	0.30259708129297

Figure 11: Difference (error) calculations between the provided TLEs and the calculated TLEs based on the Double-R method with colorscale for heatmap error (green for low error, orange and red for high error values).

	TLE B: 25165	Double-R
inc [deg]	52.016	51.9239576964323
RAAN [deg]	303.699	300.495997043193
ecc	0.0005433	0.0697757707427007
omega [deg]	319.9573	144.798765881347
n-bar [rev/day]	12.12023409	10.775341727715
Semi-major axis [km]	8120.03551291501	8658.59938178775
Period [h]	2.02276585208223	2.22730755148769

Figure 12: Comparison between the Double-r iteration calculation and the most likely TLE candidate. Note that inc and RAAN are very close, as is expected. Eccentricity is off, but both are very circular. Semi major axis and argument of perigee values are off, but not surprisingly, due to the circularity of the orbit.

Problem 3

Lambert's Problem Solutions Comparison.
Using universal variable method, Gauss' method, Izzo/Gooding method, and minimum energy, find and compare the velocity vectors for the orbit give two positions, a difference in time, and going the short way around. Please compare the COES as well as the vectors. Discuss why the answers vary.

```
r0 vect = 15,945.34 (Ihat)(km)
r1 vect = 12,214.83899 (Ihat) +  10,249.46731 (Jhat)(km)
tm = short way
deltat = 76.0 mins
```

Solution.

See Appendix C for results and code for Problem 3

Four methods for solving Lambert's problem were employed for the initial orbit determination for two position vectors (given in ECI) and a time between the two observations. No simplifying assumptions were made, since these vectors lie in a plane (no z term). The universal variable, Gauss, and minimum energy algorithms were a blend of homemade and AERO557-provided functions, while the Izzo-Gooding function was retrieved from `MATLAB` Central. All calculations were validated against Vallado's Example 7.5 (third edition). Results are shown in Figs. 13 - 16.

Why the differences?

As in Problem 2, the Gauss method is useful when angular separation is less than 10°, and especially when the extended mode is used. The initial position vectors have an angular separation of $\Delta\nu = 40°$. This is one reason why the Gauss method is not as accurate as the other methods.

The Universal Variable and Izzo-Gooding methods performed nearly identically; at least to the third decimal (sometimes the fourth) in velocities and very close in the COEs. According to [2], numerical analysis indicates that the universal-variable and Battin approaches [not studied here] provide the most accurate answer. The Izzo-Gooding method, a relatively new method, claims robustness and accuracy with two algorithms in one code. In this study we assumed a zero-rev solution; both position vectors were obtained in a single pass.

Finally, the minimum energy approach was used to determine not only the velocity vectors (using a Lambert's solver inside the function after computing the minimum energy state), but also the *minimum energy* semi major axis and minimum energy transfer time, Δt_m, between the two position vectors. This minimum energy transfer time was compared with the parabolic solution (non-elliptical) and the transfer time is given in Figs. 15 and 16. Note that the minimum energy transfer time is slightly lower (75.67 minutes) than the given transfer time (76 minutes); this is a coincidence.

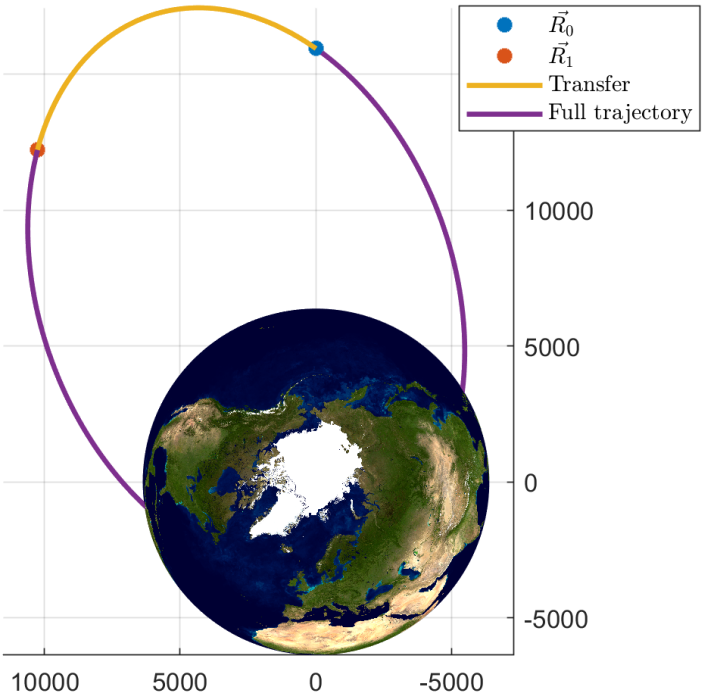

Figure 13: Solution to Lambert's problem for the two given position vectors. Since the trajectory crashes into earth, any number of speculative ideas may be made about the origin and nature of the observed object. This trajectory is confirmed using the four IOD methods and is validated against the example in [2].

```
*** Gauss Method
V0 (Gauss Method) is: 3.5705 km/s
V1 (Gauss Method) is: 3.5705 km/s

*** Universal Variable Method
V0 (Universal Variable Method) is: 3.5696 km/s
V1 (Universal Variable Method) is: 3.5696 km/s

*** Izzo-Gooding Method
V0 (Izzo-Gooding Method) is: 3.5696 km/s
V1 (Izzo-Gooding Method) is: 3.5696 km/s
*** Minimum Energy Method
V0 (Minimum Energy Method) is: 3.5695 km/s
V1 (Minimum Energy Method) is: 3.5695 km/s
```

Figure 14: Speeds obtained from the four IOD methods explored

Finally, the COEs were compared. All methods agreed fairly reasonalby (Gauss being the worst), and all gave 0° inclinations and unsolvable RAAN and argument of perigee values. For an equatorial planar orbit, Ω and ω are indeed unsolvable, since no node line exists between two parallel planes.

	Gauss Method	Universal Variable	Izzo-Gooding Method	Minimum Energy Method
v0x [km/s]	2.05994333876808	2.05888373419334	2.05891074466021	2.04738185098226
v0y [km/s]	2.91633934210809	2.91598259490204	2.91596375936091	2.92401986811616
v0z [km/s]	0	0	0	0
Trajectory Time, h	76	76	76	75.6708777398771
v1x [km/s]	-3.4525924722187	-3.45155388135579	-3.45156246532114	-3.44790918737414
v1y [km/s]	0.909941481474434	0.910347262312584	0.910315471457215	0.923897438948356
v1z [km/s]	0	0	0	0

Figure 15: Velocity vectors associated with the two known position vectors obtained from four IOD methods

	Gauss Method	Universal Variable	Izzo-Gooding Method	Minimum Energy Method
h [km2/s]	46502.0223652899	46496.3339097953	46496.0335706879	46624.4909638674
inc [deg]	0	0	0	0
raan [deg]	NaN	NaN	NaN	NaN
ecc	0.702175159721244	0.702201150091742	0.702205826390994	0.700202980087835
w [deg]	NaN	NaN	NaN	NaN
a [km]	10701.4153631316	10699.5677438606	10699.5681389674	10699.4838536717

Figure 16: COEs obtained from the four IOD methods

References

[1] Chris Peat. *Frequently Asked Questions*. Heavens-Above, 2024.

[2] David A Vallado. *Fundamentals of astrodynamics and applications*, volume 12. Springer Science & Business Media, 2001.

Appendix A: Problem 1 Script

Table of Contents

..... 1
Problem 1: Topocentric Coordinates 1
Find r, v of spacecraft at epoch 2
Propagate object forward to mission start time (08:00:00 Jan 11 UTC; 00:00:00 Jan 11 SLO time) 3
Visualization 4
Mission: Propagate 24 hours from Jan 11 to Jan 12 (SLO time); check viewing angles 4
Extract data and filter unwanted data 7
Plot Eclipse times 10


```
%{

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Winter 2024: AERO 557 | Advanced Orbital Mechanics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

HW #1
%}
tic
% housekeeping
clear all; close all; clc;
addpath("C:\MATLAB_CODE\Orbits\")

disp("HW 1, PROBLEM 1 (Topocentric Coordinates)")
disp(" ")
```

Problem 1: Topocentric Coordinates

```
%{
Hubble Space Telescope TLEs:

TLE nomenclature:
Line #  Sat #    Init. desig.    Epoch           Stuff   ~~~
1       20580U  90037B          24002.44611677  .00005755 00000-0 28686-3 0
9993

    ~       inc.    RAAN    ecc     arg. perig  Mean Anom.  mean motion
~     ~
2   20580   28.4707 20.1793 0002360 26.9978     87.2701     15.15398748
65169 2

Initial:
1 20580U 90037B 24002.44611677 .00005755 00000-0 28686-3 0 9993
2 20580 28.4707 20.1793 0002360 26.9978 87.2701 15.15398748651692

Updated:
1 20580U 90037B 24010.28323428 .00005912 00000-0 29430-3 0 9991
2 20580 28.4707 327.8603 0002613 104.9653 10.4281 15.15476605652889
```

```
%}

% PREDICT THE LOOK ANGLES (right asc./decl and Az./El.) AND associated
% times for January 11 (in SLO)

% IN SLO time: "Jan 11"         = 00:00:00 Jan 11 -> 23:59:59 Jan 11
% In UTC time: "Jan 11 [SLO]"   = 08:00:00 Jan 11 -> 07:59:59 Jan 12

%{
SLO COORDINATES
35.3540 north (of the equator)
120.3757 west (of the prime meridian)
105.8 m altitude
%}

% Given position of SLO
longitude = -120.3757;
phi = 35.3540; % geodetic latitude in radians
H = 0.1058; % km; given

% Constants
re = 6378; % radius of earth, equatorial; km
mu = 398600; % km3/s2
Re = 6378; % km
Rp = 6357; % radius of earth, polar; km
f = (Re - Rp) / (Re); % flatness factor

% EPOCH: 10 January 2024 06:47:51
% **This is TLE time. Need to propagate forward to mission start time.**
```

Find r, v of spacecraft at epoch

```
raan = deg2rad(327.8603);   % rad
inc = deg2rad(28.4707);     % rad
w = deg2rad(104.9653);      % rad
ecc = 0.0002613;
Me = deg2rad(10.4281);
E = newtonsKepler(ecc, Me);
TA = 2*atan((tan(E/2))/(sqrt((1 - ecc)/(1 + ecc)))); % rad

% Find h
% mean motion = n
n = 2*pi*15.15476605/86400; % 1/s
T = 2*pi/n;
a = (sqrt(mu)/n)^(2/3);
h = sqrt( a * mu * (1-ecc^2)   );  % km2/s
disp("h is: " + h + " km2/s (good!)")

% find ra, rp
ra = h^2/mu * (1/(1-ecc^2))*ecc + a;
rp = 2*a - ra;
checkecc = (ra - rp) / (ra + rp);
checkecc = ecc - checkecc; % good
```

```
COEs = [raan,inc,w,h,ecc,TA];
[r.epoch,v.epoch] = rv_from_COESNew(COEs);
```

Propagate object forward to mission start time (08:00:00 Jan 11 UTC; 00:00:00 Jan 11 SLO time)

```
% Start mission (Jan 11 SLO time)
y = 2024;
m = 1;
d1 = 11; % start mission day (UTC)
d2 = 12; % finish mission day (UTC)
UT1 = [08 00 00]; % start mission UTC time
UT2 = [07 59 59]; % finish mission UTC time

% Calculate seconds between two dates (UTC)
t.epoch = datetime('10-Jan-2024 06:47:51');
t.missionstart = datetime('11-Jan-2024 08:00:00');
t.missionfinish = datetime('12-Jan-2024 07:59:59');

%.... How long to propagate orbit for (epoch --> mission start time)
dt = t.missionstart - t.epoch;
t0 = seconds(dt); % time from epoch to mission t0

dt2 = t.missionfinish - t.missionstart;
tmission = seconds(dt2); % 24 hours of mission time (less 1 second)

%..... ode45 options
options = odeset('RelTol', 1e-8, 'AbsTol',1e-8);
tspan = [0 t0]; % from epoch to mission start time
state.epoch = [r.epoch;v.epoch]; % r,v at epoch; ECI

%.....Call ode (NO PERTURBATIONS) EPOCH TO MISSION START TIME
[timenew.epoch, statenew.epoch] = ode45(@pig,tspan,state.epoch,options,mu);

% Osculating (no perts) orbit after epoch->mission start time
r.osc = statenew.epoch(:,1:3);
v.osc = statenew.epoch(:,4:6);
r.missionstart = r.osc(end,:)';
v.missionstart = v.osc(end,:)';

% Run ode again for entire mission for visualization
state.mission = [r.missionstart; v.missionstart];

%.....Call ode (NO PERTURBATIONS) FULL MISSION
tspan = [0 tmission];
[timenew.mission, statenew.mission] =
ode45(@pig,tspan,state.mission,options,mu);

r.mission = statenew.mission(:,1:3);
```

```
v.mission = statenew.mission(:,4:6);

% Check UV (it is good)
TOL = 1e-8;
deltat = 1;
Ruv = r.epoch;
Vuv = v.epoch;
for i = 1:t0
[Ruv,Vuv] = universalVar(mu,deltat,Ruv,Vuv,TOL);
end
```

Visualization

```
% Plot earth and orbit (epoch -> mission start) in ECI for check
figure()
% Earth
   % h1 = gca;
   % earth_sphere(h1)
   % grid on
   % hold on
   hold on
opts.Units = 'km';
opts.RotAngle = 175; % by trial and error to get SLO under the path (globe
is just an image; not in position at epoch)
units = opts.Units;
planet3D('Earth',opts);
grid on
% Osculating (no perts) orbit
plot3(r.osc(:,1),r.osc(:,2),r.osc(:,3),'r','LineWidth',1)
% Position at EPOCH
plot3(r.osc(1,1), r.osc(1,2), r.osc(1,3),'r*','LineWidth',2)

% Position at MISSION START
plot3(r.osc(end,1),r.osc(end,2),r.osc(end,3),'mx','LineWidth',2)

% Orbit over mission time (jan 11 SLO time)
plot3(r.mission(:,1),r.mission(:,2),r.mission(:,3),'b','LineWidth',1)
hold off

legend('','Osculating (no perts) orbit','Epoch start (Jan 10, 2024, 22:47:51
SLO time)',...
    'Mission start (Jan 11, 2024, 00:00:00 SLO time)','Mission trajectory
(24h; no perts)','Location','best')
```

Mission: Propagate 24 hours from Jan 11 to Jan 12 (SLO time); check viewing angles

```
%{
Plan:
    Propagate orbits forward (2-body; no perts) over timespan.
    Find LST at each timestep
```

```
    Propagate JD
    Get new Rs
    Get new rho
    Get new RA and dec
    Get new az, el
    Check eclipse
%}

% Create loop for calculating all items
y = 2024;
m = 1;
d = 11; % start mission day (UTC)
UT = [08 00 00]; % start mission UTC time

dt = 1; % update every 1 s
TOL = 1e-8;
R = r.missionstart;
V = v.missionstart;
n = 86400; % number of iterations (24 hours in seconds)

% Set up ECI frame
Ihat = [1 0 0];
Jhat = [0 1 0];
Khat = [0 0 1];

% Initial jd
[jd, ~, ~] = juliandateFunction(y,m,d,UT);

% Preallocate vectors
nu.site = zeros(n,1);
nu.sc = nu.site;
az = nu.site;
el = nu.site;
aer = zeros(n,3);
range = nu.site;
JD = [];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% start big loop
 for i = 1:n % total mission time, s

    %..... Get JD and local sidereal time (LST)
    % Need to update y, m, d, UT here
    tt = datetime(jd,'ConvertFrom','juliandate');
    DateVector = datevec(tt);
    y = DateVector(1);
    m = DateVector(2);
    d = DateVector(3);
    UT = DateVector(4:6);

    % Find location of sun in ECI frame (code courtesy of Curtis)
    [~,~, R_Sun] = solar_position(jd);    % output in km

    % Find if s/c is in eclipse (nu = 0, BAD) or in the sun (nu = 1, GOOD)
    nu.sc(i) = checkeclipse(R,R_Sun,re);
```

```
    %..... Solve for Rsite
    theta = local_sidereal_time_justin(y,m,d,UT,longitude); % degrees
    K_term = ( (Re * (1-f)^2 ) / (sqrt(1 - (2*f - f^2)*sind(phi)^2)  ) +
H)*sind(phi)*Khat;
    Rs = ((Re / (sqrt(1 - (2*f - f^2) * sind(phi)^2 ) ) ) + H) * cosd(phi) *
(cosd(theta) * Ihat + sind(theta) * Jhat) + K_term;

    %..... Check if Rsite is in light or dark
    dotRsunRs = dot(R_Sun,Rs);
    if dotRsunRs > 0        % site in light; BAD
        nu.site(i) = 0;
        else
        nu.site(i) = 1; % site in dark; GOOD
    end

    %%%%%%%%%%%% Run Dr. A Code? %%%%%%%%%%%%%%%%
    drA = 0;

    if drA == 1
    % DR ABERCROMBY CODE
    %..... Perform viewing angle calculations
    % Find RA and dec in ECI
    dec = asin(R(3)/norm(R)); % rad
    l = R(1)/norm(R);
    m = R(2)/norm(R);

    % Get quadrant right for RA
    if m > 0
        RA = acos(l/cos(dec)); % rad
        else
        RA = 2*pi - acos(l/cos(dec)); % rad
    end

    % Find rho
    rho = R-Rs;
    % .... Call Dr. A function for elevation and azimuth
    lat = 35.3540;
    lon = -120.3757;
    [el,az] = r2elaz(rho,lat,lon,theta); % degs


    breakpoint = 1;


    % end dr A code part
    else
        % MATLAB CODE
        %..... Find az, el using MATLAB funct
        lla = [35.3540,-120.3757,105.8];
        Rsc = R'.*1000; % m

        % call az, el, range funct (matlab)
        aer(i,:) = eci2aer(Rsc,DateVector,lla);
```

```
    end % dr A code

    %..... Update time
    jd = jd + 1/n; % new jd = old jd + one time step
    T = datetime(jd,'ConvertFrom','juliandate');

    % store jd
    JD(i) = jd;

    %..... Propagate orbit forward one time step using UV
    [R,V] = universalVar(mu,dt,R,V,TOL);

 end % big loop %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% end loop
toc
```

Extract data and filter unwanted data

```
if drA ~=1 % if NOT running Dr A code; otherwise az, el already defined
    az = aer(:,1);
    el = aer(:,2);
    range = aer(:,3);
end

% Find indices in az, el when spacecraft is in sunlight and when site is in
darkness
bothTrue = zeros(n,1);
for i = 1:n
    bothTrue(i) = nu.sc(i) + nu.site(i);
    if bothTrue(i) ~= 2
        az(i) = NaN;
        el(i) = NaN;
    end
end

% Find only positive elevations
for i = 1:n
    if el(i) < 0
        el(i) = NaN;
    end
end

figure
plot((1:n),el,'LineWidth',2)
hold on
findpeaks(el) % disp peaks

% Graph pretty
ylim padded
xlim padded
xLab = xlabel('Mission time, sec','Interpreter','latex');
yLab = ylabel('Elevation, degrees','Interpreter','latex');
plotTitle = title('[Filtered] HST visible elevation by time from SLO, no
masking','interpreter','latex');
```

```
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 12)
grid on
legend('Elevation, degrees','Peaks', 'interpreter','latex','Location',
'best')

% Find locations of start, peak, and stop
loc.view1.start = 61896;
loc.view1.peak = 62187;
loc.view1.stop = 62479;

loc.view2.start = 67854;
loc.view2.peak = 68202;
loc.view2.stop = 68223;

loc.view3.start = 73869;
loc.view3.stop = 73925;

% Find elevation at start, peak, stop
view1.el.start = el(loc.view1.start);
view1.el.peak = el(loc.view1.peak);
view1.el.stop = el(loc.view1.stop);

view2.el.start = el(loc.view2.start);
view2.el.peak = el(loc.view2.peak);
view2.el.stop = el(loc.view2.stop);

view3.el.start = el(loc.view3.start);
view3.el.stop = el(loc.view3.stop);

% Find azimuth at start, peak, stop
view1.az.start = az(loc.view1.start);
view1.az.peak = az(loc.view1.peak);
view1.az.stop = az(loc.view1.stop);

view2.az.start = az(loc.view2.start);
view2.az.peak = az(loc.view2.peak);
view2.az.stop = az(loc.view2.stop);

view3.az.start = az(loc.view3.start);
view3.az.stop = az(loc.view3.stop);

% ..... Find times of peaks
% View 1
time.view1.start = JD(loc.view1.start);
time.view1.peak = JD(loc.view1.peak);
time.view1.stop = JD(loc.view1.stop);
view1.date.start
= datetime(time.view1.start,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');
view1.date.peak
```

```
= datetime(time.view1.peak,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');
view1.date.stop
= datetime(time.view1.stop,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');

% View 2
time.view2.start = JD(loc.view2.start);
time.view2.peak = JD(loc.view2.peak);
time.view2.stop = JD(loc.view2.stop);
view2.date.start
= datetime(time.view2.start,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');
view2.date.peak
= datetime(time.view2.peak,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');
view2.date.stop
= datetime(time.view2.stop,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');

% View 3 (less than 5 deg)
time.view3.start = JD(loc.view3.start);
time.view3.stop = JD(loc.view3.stop);
view3.date.start
= datetime(time.view3.start,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');
view3.date.stop
= datetime(time.view3.stop,'ConvertFrom','juliandate','TimeZone','America/
Los_Angeles');

% .... display datetimes
viewtable = table(...
    [view1.date.start;view1.date.peak;view1.date.stop],...
    [view2.date.start;view2.date.peak;view2.date.stop],...
    [view3.date.start;view3.date.stop;view3.date.stop],...
    'VariableNames',{'View 1','View 2','View 3'},'RowName',
{'Start','Peak','Stop'});
disp(viewtable)

% display azimuth
viewtable = table(...
    [view1.az.start;view1.az.peak;view1.az.stop],...
    [view2.az.start;view2.az.peak;view2.az.stop],...
    [view3.az.start;view3.az.stop;view3.az.stop],...
    'VariableNames',{'View 1','View 2','View 3'},'RowName',{'Start az
[deg]','Peak az [deg]','Stop az [deg]'});
disp(viewtable)

% display az DIRECTIONS
viewtable = table(...
    [{'SSW'};{'SSE'};{'ESE'}],... % v1 start; v1 peak; v1 stop ... and so on
    [{'SW'};{'S'};{'SSE'}],...
    [{'WSW'};{'WSW'};{'WSW'}],...
    'VariableNames',{'View 1','View 2','View 3'},'RowName',{'Start az
```

```
direction','Peak az direction','Stop az direction'});
disp(viewtable)

% display elevations
viewtable = table(...
    [view1.el.start;view1.el.peak;view1.el.stop],...
    [view2.el.start;view2.el.peak;view2.el.stop],...
    [view3.el.start;view3.el.stop;view3.el.stop],...
    'VariableNames',{'View 1','View 2','View 3'},'RowName',{'Start el
[deg]','Peak el [deg]','Stop el [deg]'});
disp(viewtable)

% VIEW 2 MATCHES VERY CLOSELY.
% discussion to include: perts; H-A masking (?). Super close though.
% Also discuss view 1 and 3 (3 is so low it got masked out, likely).
```

Plot Eclipse times

```
figure
plot(1:n,nu.site,'*')
hold on
plot(1:n,nu.sc,'*')
posL = [0 0.9 2.65e4 0.2];
rectangle('Position',posL,'EdgeColor','k','LineWidth',2,'LineStyle','-') %[x
y w h]

posR = [6.1e4 0.9 2.59e4 0.2];
rectangle('Position',posR,'EdgeColor','k','LineWidth',2,'LineStyle','-') %[x
y w h]
plot(6.1e4,0.9,'-k','LineWidth',2) % just for the key

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Mission time, s','Interpreter','latex');
yLab = ylabel('Viewing conditions (met/not met)','Interpreter','latex');
plotTitle = title('Viewing times from SLO','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 12)
grid on
legend('Viewing site (1 = in dark, 0 = in sun)','Spacecraft (1
= in sun; 2 = in dark)','SC in light, site in darkness (2x)',
'interpreter','latex','Location', 'best')
```

Published with MATLAB® R2023b

Appendix B: Problem 2 Script

Table of Contents

..... 1
Setup 2
Validate Gauss NONEXTENDED code with Vallado example 7-2; page 443 2
Gauss Extended 3
Double-R 3
Print results 4
Compare COEs from state vectors by each method) 4
Print TLE comparison table 5
Double R and chosen TLE (B) compare 6


```
%{

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Winter 2024: AERO 557 | Advanced Orbital Mechanics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

HW #1 Problem #2
%}

% housekeeping
clear all; close all; clc;
addpath("C:\MATLAB_CODE\Orbits\")
mu = 398600; % km3/s2 Grav Param

disp("HW 1, PROBLEM 2 (Angles Only Observations)")
disp(" ")

%{
PROBLEM STATEMENT:
Calculate the position and velocity vectors from the double-r method (posted
code for you to
use) and Gauss' method (non-extended and extended) for topocentric angles
only observations.

In addition, compare the resulting COEs from the state vectors as well.

Discuss the differences
and thoughts on each of the methods.

Please note, in the future, the Gauss method is **always** run with the
extension.

Finally, pick what TLE object might be close for this object (assume no
perturbations) and just compare the COES.

Observation RA (degs)    DEC (degs)  Date (UT)    Time (UT)
1           30.381       23.525      03-25-2013  03:10:30.032
2           65.134       0.774       03-25-2013  03:15:20.612
3           99.976       -30.44      03-25-2013  03:20:32.777
```

```
Observations are taken at Cal Poly Observatory 35.30 north, 120.66 west,
105.8 m altitude

TLE options:
(A)
1 25623U 99004C 13109.04882318 -.00000094 00000-0 -16690-3 0 1204
2 25623 051.9974 067.7982 0012092 184.1231 215.4516 11.86494224651645

(B)
1 25165U 98008D 13083.14572197 -.00000211 00000-0 -12941-2 0 4434
2 25165 052.0160 303.6990 0005433 319.9573 182.6276 12.12023409691559

(C)
1 25946U 99058D 13104.16396495 -.00000071 00000-0 19175-3 0 939
2 25946 051.9981 329.5396 0000986 149.5293 353.4996 12.46940793622716
%}
```

Setup

Calculate position and velocity vectors using Gauss method lets start here. (follow lecture + vallado textbook helpful)

```
% Knowns; prepare to call Gauss function
lat = 35.30; % degrees N
lon = -120.66; % or + 120.66 W
altSite = 105.8; % altitude of site, meters
RA = [30.381; 65.134; 99.976]; % deg
dec = [23.525; 0.774; -30.44]; % deg
extended = 'no';

% Observation times
t1 = datetime('2013-03-25  03:10:30.032','InputFormat','yyyy-MM-dd
HH:mm:ss.SSS');
t2 = datetime('2013-03-25  03:15:20.612','InputFormat','yyyy-MM-dd
HH:mm:ss.SSS');
t3 = datetime('2013-03-25  03:20:32.777','InputFormat','yyyy-MM-dd
HH:mm:ss.SSS');

time = [datevec(t1);datevec(t2);datevec(t3)];
```

Validate Gauss NONEXTENDED code with Vallado example 7-2; page 443

```
%{
Observation 1: 11:40:00.00; alpha1 = -0.4172870; dec1 = 17.4626616
Observation 2: 11:50:00.00; alpha2 = 55.0931551; dec2 = 36.5731946
Observation 3: 12:20:00.00; alpha3 = 134.2826693; dec3 = 12.0351097
%}
%{
RA = [-0.4172870; 55.0931551; 134.2826693 ];
dec = [17.4626616; 36.5731946; 12.0351097 ];
```

```
lat = 40;
lon = -110;
altSite = 2000; % leave in meters

% Observation times (VALLADO EXAMPLE)
t1 = datetime('2007-08-20  11:40:00.00','InputFormat','yyyy-MM-dd
HH:mm:ss.SSS');
t2 = datetime('2007-08-20  11:50:00.00','InputFormat','yyyy-MM-dd
HH:mm:ss.SSS');
t3 = datetime('2007-08-20  12:20:00.00','InputFormat','yyyy-MM-dd
HH:mm:ss.SSS');
time = [datevec(t1);datevec(t2);datevec(t3)];
extended = 'no';
%}

% Call function
[r1,r2,r3,v2,rhohatVect,tau1,tau3,RsVect] =
IOD_Gauss(lat,lon,altSite,RA,dec,time,extended);
%^^ Above method validated against Vallado example 7-1 (third ed); solid!
```

Gauss Extended

Calculate position and velocity vectors using Gauss EXTENDED method

```
% Call function with EXTENDED MODEL
[r1E,r2E,r3E,v2E,~,~,~,~] = IOD_Gauss(lat,lon,altSite,RA,dec,time);


% In addition, compare the resulting COEs from the state vectors as well.
% Discuss the differences and thoughts on each of the methods.
% TO DO <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

Double-R

Calculate position and velocity vectors using Double-R method

```
% (re)define inputs
Rs1 = RsVect(:,1);
Rs2 = RsVect(:,2);
Rs3 = RsVect(:,3);
qhat1 = rhohatVect(:,1);
qhat2 = rhohatVect(:,2);
qhat3 = rhohatVect(:,3);

tau1 = tau1*86400; % Jd to seconds
tau3 = tau3*86400;

% Call Double-R
[r2RR, v2RR] = AERO557doubleR(qhat1,qhat2,qhat3,Rs1,Rs2,Rs3,tau1,tau3);
```

Print results

```
format long g
disp(" ")
q2State = table([r2;v2],[r2E;v2E],[r2RR;v2RR],'VariableNames',
{'Gauss','Gauss Extended','Double-R'},'RowName',{'r2x [km]','r2y [km]','r2z
[km]','v2x [km/s]','v2y [km/s]','v2z [km/s]'});
disp(q2State)

%..... Find COEs and compare.
% Gauss
[h.g, inc.g, RAAN.g, ecc.g, w.g, TA.g, epsilon.g, a.g, T.g, p.g] =
rv2COEs(r2,v2,mu);
% Gauss Extended
[h.ge, inc.ge, RAAN.ge, ecc.ge, w.ge, TA.ge, epsilon.ge, a.ge, T.ge, p.ge] =
rv2COEs(r2E,v2E,mu);
% Double-R
[h.rr, inc.rr, RAAN.rr, ecc.rr, w.rr, TA.rr, epsilon.rr, a.rr, T.rr, p.rr] =
rv2COEs(r2RR,v2RR,mu);
```

Compare COEs from state vectors by each method)

```
disp(" ")
disp("Angles-Only IOD COE comparison")
q2State = table([h.g;inc.g;RAAN.g;ecc.g;w.g;TA.g;a.g;T.g],
[h.ge;inc.ge;RAAN.ge;ecc.ge;w.ge;TA.ge;a.ge;T.ge],...
    [h.rr;inc.rr;RAAN.rr;ecc.rr;w.rr;TA.rr;a.rr;T.rr],...
   'VariableNames',{'Gauss','Gauss Extended','Double-R'},'RowName',{'h [km2/
s]','inc [deg]','RAAN [deg]','ecc','omega [deg]',...
   'True anomaly [deg]','Semi-major axis [km]','Period [s]'});
disp(q2State)

% Compute other orbital parameters for the RR method
n.rr = 2*pi / T.rr;
T.rr = T.rr / 3600; % in hours now.

% Find COEs for given TLEs and compare.
%{
(A)
1 25623U 99004C 13109.04882318 -.00000094 00000-0 -16690-3 0 1204
2 25623 051.9974 067.7982 0012092 184.1231 215.4516 11.86494224651645

(B)
1 25165U 98008D 13083.14572197 -.00000211 00000-0 -12941-2 0 4434
2 25165 052.0160 303.6990 0005433 319.9573 182.6276 12.12023409691559

(C)
1 25946U 99058D 13104.16396495 -.00000071 00000-0 19175-3 0 939
2 25946 051.9981 329.5396 0000986 149.5293 353.4996 12.46940793622716
%}
```

```
% TLE (A) [degrees]
A.inc = 51.9974;
A.RAAN = 067.7982;
A.ecc = 0.0012092;
A.w = 184.1231;
A.Me = 215.4516;
A.nbar = 11.86494224; %rev/day
A.n = A.nbar*2*pi*86400^-1; % convert to rad/sec
A.T = 2*pi / (A.n); % T = 2pi/n, n in rad/sec
A.T = A.T / 3600; % into hours
A.a = (mu/A.n^2)^(1/3);

% TLE (B) [degrees]
B.inc = 52.0160;
B.RAAN = 303.6990;
B.ecc = 0.0005433;
B.w = 319.9573;
B.Me = 182.6276;
B.nbar = 12.12023409; %rev/day
B.n = A.nbar*2*pi*86400^-1; % convert to rad/sec
B.T = 2*pi / (B.n); % T = 2pi/n, n in rad/sec
B.T = B.T / 3600; % into hours
B.a = (mu/B.n^2)^(1/3);

% TLE (C) [degrees]
c.inc = 51.9981;
c.RAAN = 329.5396;
c.ecc = 0.0000986;
c.w = 149.5293;
c.Me = 353.4996;
c.nbar = 12.46940793; %rev/day
c.n = c.nbar*2*pi*86400^-1; % convert to rad/sec
c.T = 2*pi / (c.n);
c.T = c.T / 3600; % into hours
c.a = (mu/c.n^2)^(1/3);

c.T2 = 2*pi / (c.n);
c.a2 = (mu * (c.T2/(2*pi))^2 ) ^ (1/3);
```

Print TLE comparison table

```
disp(" ")
disp("Given TLE data to compare")
q2State = table([A.inc;A.RAAN;A.ecc;A.w;A.nbar;A.a;A.T],
[B.inc;B.RAAN;B.ecc;B.w;B.nbar;B.a;B.T],
[c.inc;c.RAAN;c.ecc;c.w;c.nbar;c.a;c.T],...
   'VariableNames',{'A','B','C'},'RowName',{'inc
[deg]','RAAN [deg]','ecc','omega [deg]','n-bar [rev/day]',...
   'Semi-major axis [km]','Period [h]'});
disp(q2State)

% Find relative error between double-r method (best) and given TLEs.
```

```
errA = abs([inc.rr - A.inc; RAAN.rr - A.RAAN; ecc.rr
- A.ecc; w.rr - A.w; n.rr - A.n; a.rr - A.a; T.rr - A.T]); % T in hours
errB = abs([inc.rr - B.inc; RAAN.rr
- B.RAAN; ecc.rr - B.ecc; w.rr - B.w; n.rr - B.n; a.rr - B.a; T.rr - B.T]);
errC = abs([inc.rr - c.inc; RAAN.rr
- c.RAAN; ecc.rr - c.ecc; w.rr - c.w; n.rr - c.n; a.rr - c.a; T.rr - c.T]);
err.rr = [errA,errB,errC];

disp("Error between Double-R method and Given TLEs [abs(RR - TLE)]:")
errortable = table(errA,errB,errC,'VariableNames',
{'A', 'B', 'C'},'RowName',...
    {'Err: inc [deg]','Err: RAAN [deg]','Err: ecc','Err:
omega [deg]','Err: n-bar [rev/day]','Err: Semi-major axis [km]',...
    'Err: Period [h]'});
disp(errortable)

figure()
x = 1:7;
semilogy(x,errA,'-x',x,errB,'-o',x,errC,'-o','LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xticks(x) % location of x-ticks
xticklabels({'inc','RAAN','ecc','\omega','n-bar','a','T'}) % names
%xtickangle(45) % if you want to angle the tick labels!
%xLab = xlabel('x','Interpreter','latex');
yLab = ylabel('Error','Interpreter','latex');
plotTitle = title('Error
between Double-R Method and Given TLEs','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set(yLab,'FontSize', 14)
grid on
legend('A','B','C', 'interpreter','latex','Location', 'best')

% Go with TLE B due to how close RAAN is since inc is very close. RAAN and
% inc are connected; but since the orbit is close to circ; arg. of perigee
% is not accurate.
```

Double R and chosen TLE (B) compare

```
disp(" ")
disp("Calculation and 25165")
state_tle = table([B.inc;B.RAAN;B.ecc;B.w;B.nbar;B.a;B.T],
[inc.rr;RAAN.rr;ecc.rr;w.rr;(n.rr*86400/2/pi);a.rr;T.rr],...
   'VariableNames',{'TLE B: 25165','Double-R'},'RowName',{'inc [deg]','RAAN
[deg]','ecc','omega [deg]','n-bar [rev/day]',...
   'Semi-major axis [km]','Period [h]'});
disp(state_tle)
```

Published with MATLAB® R2023b

Appendix C: Problem 3 Script

Table of Contents

..... 1
Gauss Method 2
Universal Variable Method Adapted from [1] 3
Izzo-Gooding Method [2] 3
Minimum Energy Method 3
Display tabulated results 4
References 4


```
%{

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Winter 2024: AERO 557 | Advanced Orbital Mechanics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

HW #1 Problem #3
%}

% housekeeping
clear all; close all; clc;
addpath("C:\MATLAB_CODE\Orbits\")
mu = 398600; % km3/s2, Grav Param
re = 6378; % km, radius of earth

disp("HW 1, PROBLEM 3 (Lambert's Problem Solutions Comparison)")
disp(" ")

%{
PROBLEM STATEMENT:
Problem 3: Lambert's Problem Solutions Comparison
(note this is an example problem in
Vallado so feel free to check your answers - except for the Izzo/Gooding
part). Note the
minimum energy case here is only close because the delta-t is close to the
minimum energy
time (30 points)

Using
1. universal variable method (AERO 351 homework),
2. gauss method (introduced in class),
3. Izzo/Gooding method (from MATLAB central) and (let n = 0 for this method)
4. minimum energy (introduced in class),

find and compare the
a) velocity vector for the orbit give two positions,
b) a difference in time, and
c) going the short way around.

Please compare the
i) COES as well as the
```

```
ii) vectors
iii) Discuss why the answers vary.

R0 = 15,945.34 Ihat (km)
R1 = 12,214.83899 Ihat+ 10,249.46731 Jhat (km)
tm = short way
deltat = 76.0 mins
%}

% Distance observations
R0 = [15945.34;0;0];  % km
R1 = [12214.83899; 10249.46731;0]; % km
r0 = norm(R0);
r1 = norm(R1);
deltat = 76*60; % 76 mins in seconds
tm = 1; % short way
```

Gauss Method

Call homemade function

```
[V0.gauss,V1.gauss] = lamberts_Gauss(R0,R1,deltat);

format long g
disp("*** Gauss Method")
disp("V0 (Gauss Method) is: " + norm(V0.gauss) + " km/s")
disp("V1 (Gauss Method) is: " + norm(V1.gauss) + " km/s")

% % % Plot to visualize GAUSS METHOD
figure
hold on
opts.Units = 'km';
opts.RotAngle = 175;
units = opts.Units;
planet3D('Earth',opts);
grid on

% Initial observations
plot3(R0(1),R0(2),R0(3),'*','Linewidth',2)
plot3(R1(1),R1(2),R1(3),'*','Linewidth',2)

% Propagate using Gauss velocity vectors (between the two and all the way
% around; short way)
%..... ode45 options
options = odeset('RelTol', 1e-8, 'AbsTol',1e-8);
tspan = [0 deltat]; % from epoch to mission start time
state0 = [R0;V0.gauss]; % r,v at epoch; ECI

%.....Call ode (no perts)
[timenew0, statenew0] = ode45(@pig,tspan,state0,options,mu);

% Plot r0 - r1
plot3(statenew0(:,1),statenew0(:,2),statenew0(:,3),'LineWidth',2)
```

```
% Plot r1 - r0 (around)
tspan = [0 1.4*deltat]; % from epoch to mission start time
state1 = [R1;V1.gauss]; % r,v at epoch; ECI

%.....Call ode (no perts)
[timenew1, statenew1] = ode45(@pig,tspan,state1,options,mu);

% Plot r1 around to r0
plot3(statenew1(:,1),statenew1(:,2),statenew1(:,3),'LineWidth',2)
legend('','$\vec{R_0}$','$\vec{R_1}$','Transfer','Full
trajectory','interpreter','latex','Location','best')

% NOTE THAT IT CRASHES into earth
disp("*Although these velocity vectors work for the transfer trajectory,
this orbit crashes into earth.")
disp(" ")

% % % COEs for Gauss Solution
[h.gauss, inc.gauss, RAAN.gauss, ecc.gauss, w.gauss, ~, ~, a.gauss, T.gauss,
~] = rv2COEs(R1,V1.gauss,mu);
```

Universal Variable Method Adapted from [1]

Call lamberts with known position vectors

```
[V0.uv,V1.uv] = lamberts_universalVar(R0,R1,deltat,tm);
[h.uv, inc.uv, RAAN.uv, ecc.uv, w.uv, ~, ~, a.uv, T.uv, ~] =
rv2COEs(R1,V1.uv,mu);
disp("*** Universal Variable Method")
disp("V0 (Universal Variable Method) is: " + norm(V0.uv) + " km/s")
disp("V1 (Universal Variable Method) is: " + norm(V1.uv) + " km/s")
disp(" ")
```

Izzo-Gooding Method [2]

Inputs for I-G

```
tf = deltat/86400; % 76 minutes in days [time of flight]
m = 0; % how many orbits; 0 == less than one orbit

disp("*** Izzo-Gooding Method")
[V0.ig,V1.ig,~,~] = lambert_izzo_gooding(R0',R1',tf,m,mu);
[h.ig, inc.ig, RAAN.ig, ecc.ig, w.ig, ~, ~, a.ig, T.ig, ~] =
rv2COEs(R1,V1.ig',mu);
disp("V0 (Izzo-Gooding Method) is: " + norm(V0.ig) + " km/s")
disp("V1 (Izzo-Gooding Method) is: " + norm(V1.ig) + " km/s")
```

Minimum Energy Method

```
disp("*** Minimum Energy Method")
[V0.minE,V1.minE,deltatm] = lamberts_minimumEnergy(R0,R1,deltat,tm);
```

```
[h.minE, inc.minE, RAAN.minE, ecc.minE, w.minE, ~, ~, a.minE, T.minE, ~] =
rv2COEs(R1,V1.minE,mu);
disp("V0 (Minimum Energy Method) is: " + norm(V0.minE) + " km/s")
disp("V1 (Minimum Energy Method) is: " + norm(V1.minE) + " km/s")
```

Display tabulated results

```
%{
find and compare the
a) velocity vector for the orbit give two positions,
b) a difference in time, and
c) going the short way around.

Please compare the
i) COES as well as the
ii) vectors
iii) Discuss why the answers vary.

%}
disp("~~~~~~~~~~~~~~~~ COES")
disp(" ")
COEstable = table(...
    [h.uv;inc.uv;RAAN.uv;ecc.uv;w.uv;a.uv],...
    [h.gauss;inc.gauss;RAAN.gauss;ecc.gauss;w.gauss;a.gauss],...
    [h.ig;inc.ig;RAAN.ig;ecc.ig;w.ig;a.ig],...
    [h.minE;inc.minE;RAAN.minE;ecc.minE;w.minE;a.minE],...
    'VariableNames',{'Universal Variable','Gauss Method','Izzo-Gooding
Method','Minimum Energy Method'},...
    'RowName',{'h [km2/s]','inc [deg]','raan [deg]','ecc','w [deg]','a
[km]'});
disp(COEstable)

disp("~~~~~~~~~~~~~~~~ Velocity vectors and time difference (short way
around)")
disp(" ")
veloctable = table(...
    [V0.gauss(1);V0.gauss(2);V0.gauss(3);deltat/
60;V1.gauss(1);V1.gauss(2);V1.gauss(3)],...
    [V0.uv(1);V0.uv(2);V0.uv(3);deltat/60;V1.uv(1);V1.uv(2);V1.uv(3)],...
    [V0.ig(1);V0.ig(2);V0.ig(3);deltat/60;V1.ig(1);V1.ig(2);V1.ig(3)],...
    [V0.minE(1);V0.minE(2);V0.minE(3);deltatm/
60;V1.minE(1);V1.minE(2);V1.minE(3)],...
    'VariableNames',{'Universal Variable','Gauss Method','Izzo-Gooding
Method','Minimum Energy Method'},...
    'RowName',{'v0x [km/s]','v0y [km/s]','v0z [km/s]','Trajectory Time,
h',...
    'v1x [km/s]','v1y [km/s]','v1z [km/s]'});
disp(veloctable)
```

References

```
%{
1. Curtis, H. D. (2020). Orbital Mechanics for Engineering Students: Revised
```

```
Reprint. Butterworth-Heinemann.
2. Rody Oldenhuis, https://github.com/rodyo/FEX-Lambert
%}
```

Published with MATLAB® R2023b

Functions Used (all problems)

```
function [h, inc, RAAN, ecc, w, theta, epsilon, a, T, p] =
rv2COEs(r0Vect,v0Vect,mu)

% Outputs in degrees

% Finds the Six Classical Orbital Elements
r = norm(r0Vect);
v = norm(v0Vect);
vr = dot(r0Vect,v0Vect)/r;

% h, specific angular momentum
hVect = cross(r0Vect,v0Vect);
h = norm(hVect);

% inc, inclination
inc = acos(hVect(3)/h);

while inc > pi
    inc = inc - pi;
end
while inc < 0
    inc = inc + pi;
end

inc = inc*(180/pi);

% omega, right ascension of ascending node (RAAN)
NVect = cross([0;0;1],hVect);
N = norm(NVect);

if NVect(2) < 0
    RAAN = 2*pi - acos(NVect(1)/N);
    else
    RAAN = acos(NVect(1)/N);
end

RAAN = RAAN*(180/pi);

% ecc, eccentricity
eccVect = (1/mu).*((v^2-(mu/r)).*r0Vect-r*vr.*v0Vect);
ecc = norm(eccVect);

% w, argument of perigee

if eccVect(3) < 0
    w = 2*pi - acos(dot(NVect,eccVect)/(N*ecc));
    else
    w = acos(dot(NVect,eccVect)/(N*ecc));
end

w = w*(180/pi);
```

```
% theta, true anomaly
if vr >= 0
    theta = acos(dot(eccVect,r0Vect)/(ecc*r));
    else
    theta = 2*pi - acos(dot(eccVect,r0Vect)/(ecc*r));
end

theta = theta*(180/pi);

% epsilon, specific energy
epsilon = (0.5*v^2) - (mu/r);
%if epsilon > 0
 %   epsilon = nan;
%end

    % a, semi-major axis and semiparameter, p
    if ecc ~= 1
        a = -mu / (2*epsilon);
        p = a*(1 - ecc^2);
    else
        p = h^2 / mu;
        a = Inf;
    end

% T, period
T = 2*pi*sqrt((a^3)/mu);


end
```

Published with MATLAB® R2023b

```
function [r,v] = rv_from_COESNew(COEs)


% Calcuate r,v from COEs each time step.

% % Run rv from coes function here plainly for speed
%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%
mu = 398600;

% Bring in COEs
raan = COEs(1);
inc = COEs(2);
w = COEs(3);
h = COEs(4);
ecc = COEs(5);
theta = COEs(6);

% Preallocate for speed
cr = cos(raan);
sr = sin(raan);
ci = cos(inc);
si = sin(inc);
cw = cos(w);
sw = sin(w);
ct = cos(theta);
st = sin(theta);

% determine perifocal to GEO rotation matrix
QXx =    [  cr*cw - ci*sr*sw, cw*sr + ci*cr*sw, si*sw;
          - cr*sw - ci*cw*sr, ci*cr*cw - sr*sw, cw*si;
            si*sr,            -cr*si,           ci];
QXx = QXx';

% find r and v (Perifocal)
r_PF = (((h^2)/mu)*(1/(1+ecc*ct)))*[ct;st;0];
v_PF = (mu/h)*[-st; (ecc+ct); 0];

% calc r and v relative to the geocentric reference frame
% GEO = ECI
r = QXx*r_PF;
v = QXx*v_PF;
```

Published with MATLAB® R2023b

```
function [dstate] = pig(time,state,mu)
%{

*FOR COAST PHASE ONLY*

**This function assumes that thrust is exactly parallel to initial velocity
direction.

This function is for numerical integration (i.e., plug into ode45).
NOTE: only use this function for the BURN PHASE of the non-impulsive
maneuver. For the BURN phase, use "non_impulsive_BURN"

INPUTS:
    INITIAL CONDITIONS
    state vector(1:3) = position (rx,ry,rz)
    state vector(4:6) = velocity (vx,vy,vz)

OUTPUT: New state vector after integration.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Fall 2022; Introduction to Orbital Mechanics.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%}


rx = state(1);
ry = state(2);
rz = state(3);
vx = state(4);
vy = state(5);
vz = state(6);

r_vect = [rx ry rz];
r = norm(r_vect);

v_vect = [vx vy vz];
v = norm(v_vect);

xdot = vx;
ydot = vy;
zdot = vz;
xddot = (-mu*rx)/(r^3);
yddot = (-mu*ry)/(r^3);
zddot = (-mu*rz)/(r^3);


dstate = [xdot; ydot; zdot; xddot; yddot; zddot];


end
```

Published with MATLAB® R2023b

```
%==========================================================================
%
% planet3D  Creates high-resolution renderings of the Earth and the major
% celestial bodies in our solar system for space mechanics applications.
%
%   planet3D
%   planet3D(planet)
%   planet3D(planet,opts)
%   planet_surface = planet3D(__)
%
% See also background, ground_track.
%
% Copyright © 2021 Tamas Kis
% Last Update: 2023-05-20
% Website: https://tamaskis.github.io
% Contact: tamas.a.kis@outlook.com
%
% Technical Documentation:
% https://tamaskis.github.io/files/Visualizing_Celestial_Bodies_in_3D.pdf
%
%--------------------------------------------------------------------------
%
% ------
% INPUT:
% ------
%   planet          - (OPTIONAL) (char array) 'Sun', 'Moon', 'Mercury',
%                     'Venus', 'Earth', 'Earth Cloudy', 'Earth Coastlines',
%                     'Earth Night', 'Earth Night Cloudy', 'Mars',
%                     'Jupiter', 'Saturn', 'Uranus', 'Neptune', or 'Pluto'
%                     (defaults to 'Earth Cloudy')
%   opts            - (OPTIONAL) (1×1 struct) plot options
%       • Clipping  - (char array) 'on' or 'off' (defaults to 'off')
%                       • if 'on', the surface will be "clipped" to fit the
%                         axes when zooming in
%       • Color     - (char array or 1×3 double) line color (only relevant
%                     when drawing Earth coastlines)
%                       • can be specified as a name, short name, or RGB
%                         triplet [rgb]
%       • FaceAlpha - (1×1 double) 0 for 100% transparency, 1 for 100%
%                     opacity
%       • LineWidth - (1×1 double) line width (only relevant when drawing
%                     Earth coastlines)
%       • LineStyle - (char array) line style (only relevant when drawing
%                     Earth coastlines)
%       • Position  - (3×1 double) position of planet's geometric center
%       • RefPlane  - (char array) 'equatorial' or 'ecliptic'
%       • RotAngle  - (1×1 double) rotation angle [deg]
%       • Units     - (char array) 'AU', 'ft', 'km', 'm', 'mi', or 'nmi'
%
% -------
% OUTPUT:
% -------
```

```
%   planet_surface  - (1×1 Surface) surface object defining planet
%
% -----
% NOTE:
% -----
%   • All fields of "opts" do NOT have to be defined; when a field is left
%     undefined, the rest of the plot settings are set to default values.
%   • Use the "background" function included with download to set the plot
%     background. When using "background" to set the plot background, the
%     function call on "background" must occur BEFORE the function call on
%     "planet3D", otherwise the background will be plotted over the
%     celestial body.
%   • If you want to produce separate plots on separate figures using the
%     "planet3D" function, always use the "drawnow" command before
%     initializing a new figure to ensure that the correct plots are drawn
%     on the correct figures.
%
%==========================================================================
function planet_surface = planet3D(planet,opts)

    % ----------------------------
    % Conversion factors and data.
    % ----------------------------

    % conversion factors
    factors = {'AU'   1/149597870000;
               'ft'   100/30.48;
               'km'   0.001;
               'm'    1;
               'mi'   100/160934.4;
               'nmi'  1/1852};

             % planet/body          radius,       flattening,    obliquity,
             %                      R [m]         f [-]          obl [deg]
    data = {'Sun'                   696000e3      0.000009       0;
            'Moon'                  1738.0e3      0.0012         6.68;
            'Mercury'               2439.0e3      0.0000         0.0;
            'Venus'                 6052.0e3      0.000          177.3;
            'Earth'                 6378.1363e3   0.0033528131   23.45;
            'Earth Coastlines'      6378.1363e3   0.0033528131   23.45;
            'Earth Cloudy'          6378.1363e3   0.0033528131   23.45;
            'Earth Night'           6378.1363e3   0.0033528131   23.45;
            'Earth Night Cloudy'    6378.1363e3   0.0033528131   23.45;
            'Mars'                  3397.2e3      0.00647630     25.19;
            'Jupiter'               71492.0e3     0.0648744      3.12;
            'Saturn'                60268.0e3     0.0979624      26.73;
            'Uranus'                25559.0e3     0.0229273      97.86;
            'Neptune'               24764.0e3     0.0171         29.56;
            'Pluto'                 1151.0e3      0.0            118.0};

    % -----------------------------------
    % Sets (or defaults) plotting options.
    % -----------------------------------
```

```
% defaults "planet" to 'Earth Cloudy' if not input
if (nargin == 0) || isempty(planet)
    planet = 'Earth Cloudy';
end

% sets position of planet's geometric center (defaults to origin)
if (nargin < 2) || ~isfield(opts,'Position')
    position = [0;0;0];
else
    position = opts.Position;
end

% sets rotation angle (defaults to 0)
if (nargin < 2) || ~isfield(opts,'RotAngle')
    theta = 0;
else
    theta = opts.RotAngle;
end

% sets conversion factor (defaults to 1, assuming units of m)
if (nargin < 2) || ~isfield(opts,'Units')
    units = 'm';
else
    units = opts.Units;
end

% sets reference plane (defaults to equatorial plane)
if (nargin < 2) || ~isfield(opts,'RefPlane')
    reference_plane = 'equatorial';
else
    reference_plane = opts.RefPlane;
end

% sets transparency (defaults to 1 so celestial body is solid)
if (nargin < 2) || ~isfield(opts,'FaceAlpha')
    FaceAlpha = 1;
else
    FaceAlpha = opts.FaceAlpha;
end

% determines obliquity
if strcmpi(reference_plane,'ecliptic')
    obl = data{strcmpi(data(:,1),planet),4};
else
    obl = 0;
end

% sets clipping (defaults to 'off')
if (nargin < 2) || ~isfield(opts,'Clipping')
    Clipping = 'off';
else
    Clipping = opts.Clipping;
end
```

```
% sets line color (defaults to default MATLAB color)
if (nargin < 2) || ~isfield(opts,'Color')
    Color = [0,0.4470,0.7410];
else
    Color = opts.Color;
end

% sets line style (defaults to solid line)
if (nargin < 2) || ~isfield(opts,'LineStyle')
    LineStyle = '-';
else
    LineStyle = opts.LineStyle;
end

% sets line width (defaults to 0.5)
if (nargin < 2) || ~isfield(opts,'LineWidth')
    LineWidth = 0.5;
else
    LineWidth = opts.LineWidth;
end

% --------------------------------
% Geometry of the celestial body.
% --------------------------------

% determines mean equatorial radius and flattening
R = data{strcmpi(data(:,1),planet),2};
f = data{strcmpi(data(:,1),planet),3};

% conversion factor to use
conversion_factor = factors{strcmpi(factors(:,1),units),2};

% determines semi-major and semi-minor axes of body
a = conversion_factor*R;
b = a*(1-f);

% coordinates of ellipsoid centered at (0,0,0) using 400 panels
[x,y,z] = ellipsoid(0,0,0,a,a,b,400);

% ------------------------------------------------------------
% Defining surfaces/coordinates needed to draw celestial body.
% ------------------------------------------------------------

% not drawing Earth coastlines
if ~strcmpi(planet,'Earth Coastlines')

    % loads image data
    if strcmpi(planet,'Earth Cloudy')
        cdata = imread('earth.png')+imread('clouds.png');
    elseif strcmpi(planet,'Earth Night Cloudy')
        cdata = imread('earthnight.png')+0.1*imread('clouds.png');
    else
        cdata = imread(strcat('',lower(planet),'.png'));
    end
```

```
    % draws planet
    planet_surface = surface(x,y,z,'FaceColor','texture',...
        'EdgeColor','none','CData',flipud(cdata),'DiffuseStrength',...
        1,'SpecularStrength',0,'FaceAlpha',FaceAlpha);

end

% drawing Earth coastlines
if strcmpi(planet,'Earth Coastlines')

    % white surface (lines will be plotted on top of this)
    planet_surface = surface(x,y,z,'FaceColor','w','EdgeColor',...
        'none','DiffuseStrength',1,'SpecularStrength',0,...
        'FaceAlpha',FaceAlpha);

    % loads coastline data
    coastlines_data = struct2cell(load('coastlines_data'));
    coastlines_data = [coastlines_data{:}];

    % extracts ECEF coordinates of coastlines
    x_coast = coastlines_data.X;
    y_coast = coastlines_data.Y;
    z_coast = coastlines_data.Z;

end

% -------------------
% Performs rotations.
% -------------------

% transformation matrix for rotation
R3 = [ cosd(theta)   sind(theta)   0;
      -sind(theta)   cosd(theta)   0;
       0             0             1];

% transformation matrix for tilt
R1 = [1   0            0;
      0   cosd(obl)   -sind(obl);
      0   sind(obl)    cosd(obl)];

% axes for rotations (must be row vectors)
alpha1 = [1,0,0];
alpha2 = (R1*[0;0;1])';

% tilts celestial body if referenced to ecliptic plane
rotate(planet_surface,alpha1,obl);

% rotates celestial body about its 3rd axis
rotate(planet_surface,alpha2,theta);

% rotates coordinates of coastlines
if strcmpi(planet,'Earth Coastlines')
    new_coordinates = R3*R1*[x_coast';y_coast';z_coast'];
```

```
    x_coast = new_coordinates(1,:);
    y_coast = new_coordinates(2,:);
    z_coast = new_coordinates(3,:);
end

% ---------------------
% Performs translation.
% ---------------------

% translates coordinates of surface object
planet_surface.XData = planet_surface.XData+position(1);
planet_surface.YData = planet_surface.YData+position(2);
planet_surface.ZData = planet_surface.ZData+position(3);

% translates coordinates of Earth coastlines
if strcmpi(planet,'Earth Coastlines')
    x_coast = x_coast+position(1);
    y_coast = y_coast+position(1);
    z_coast = z_coast+position(1);
end

% ----------------------------------------------------------------
% Drawing additional lines (i.e. coastlines or rings of Saturn).
% ----------------------------------------------------------------

% draws coastlines
if strcmpi(planet,'Earth Coastlines')
    hold on;
    plot3(x_coast,y_coast,z_coast,'LineWidth',LineWidth,'LineStyle',...
        LineStyle,'Color',Color);
    hold off;
end

% draws rings of Saturn
if strcmpi(planet,'Saturn')

    % reads in image
    cdata_rings = imread('saturnrings.png');

    % determines number of different colors in ring (if you look at the
    % image, the way it is formatted just looks like horizontal bands
    % of colors)
    n = size(cdata_rings,2);

    % preallocates array to store colors
    colors = zeros(n,3);

    % extracts rgb values from image data
    for i = 1:n
        colors(i,:) = cdata_rings(1,i,:);
    end

    % shrinks the data set of colors down to only 200 (this is for
    % speed - we plot the bands of Saturns rings as individual lines
```

```
        % and don't want to plot thousands of lines) - this shrinking
        % process is condensed from the reduced_data_points function (see
        % https://github.com/tamaskis/Useful_Functions_for_MATLAB-MATLAB/
        % blob/main/functions/reduce_data_points.m)
        n_new = 200;
        colors = colors(1:round(n/n_new):n,:);

        % scales colors to between 0 and 1 (currently between 0 and 255)
        colors = colors/255;

        % plots the rings
        theta = 0:0.001:2*pi;
        hold on;
        for i = 1:n_new

            % this comes from the fact that Saturns rings extend from 7000
            % to 80000 km (7,000,000 to 8,000,000 km) from the surface of
            % the planet s(https://en.wikipedia.org/wiki/Rings_of_Saturn)
            r = conversion_factor*(R+7000000+((80000000-7000000)/n_new)*i);

            % x, y, and z coordinates of Saturns rings in equatorial plane,
            % centered at origin
            x_ring = r*cos(theta);
            y_ring = r*sin(theta);
            z_ring = ones(size(theta));

            % tilts rings
            new_coordinates = R1*[x_ring;y_ring;z_ring];
            x_ring = new_coordinates(1,:);
            y_ring = new_coordinates(2,:);
            z_ring = new_coordinates(3,:);

            % centers rings on main body
            x_ring = x_ring+position(1);
            y_ring = y_ring+position(1);
            z_ring = z_ring+position(1);

            % plots the jth ring
            plot3(x_ring,y_ring,z_ring,'Color',colors(i,:));

        end
        hold off;

    end

    % ----------------------
    % Basic plot formatting.
    % ----------------------

    % set axis clipping
    ax = gca;
    ax.Clipping = Clipping;

    % equal data unit lengths along each axis
```

```
    axis equal;

    % 3D view
    view(3);

end
```

Published with MATLAB® R2023b

```
function [juliandateReturn, UT_decimal, j0] = juliandateFunction(y,m,d,UT)
%   This function outputs the julian date (JD) with inputs year, month,
%   day, and GMT (UT)
% Author: Justin Self AERO 351 Fall 2022

% y = four digit year
% m = two digit month
% d = two digit day
% UT = array that contains the UT in the form [hour min sec]

j0 = 367*y - floor((7*(y+floor((m+9)/12)))/4) + floor((275*m)/9) + d +
1721013.5;

% j0 = Julian date at 0 hours UT (noon)
% Now ww need to convert HH:MM:SS to HH.HH (decimal hours)

UT_decimal = UT(1) + UT(2)/60 + UT(3)/3600; % 60 min / hour, 3600 sec / hour
juliandateReturn = j0 + (UT_decimal/24);

end
```

Published with MATLAB® R2023b

```
function [LST] = local_sidereal_time_justin(y,m,d,UT,longitude)
% This function takes a longitudinal location and date and calculates local
sidereal
% time

% Convert UT vector to decimal equiv
UT_decimal = UT(1) + UT(2)/60 + UT(3)/3600; % 60 min / hour, 3600 sec / hour

% Call julian_date_justin function to obtain j0 only **not JD**
[~,~,j0] = juliandateFunction(y,m,d,UT);

j2000 = 2451545.0; % always this constant value
julian_century = 36525; % days

% Now we have j0, find t0

T0 = (j0 - j2000)/ julian_century; % unitless

% Compute Greenwich sidereal time THETA G0 at 0 hours Zulu
thetaG0 = 100.4606184 + 36000.77004 * T0 + 0.000387933 * T0^2 - 2.58 * 10^-8
* T0^2; % degrees

% Calculate Greenwich sidereal time at ANY time, theta G
thetaG = thetaG0 + 360.98564724 * (UT_decimal/24); % degrees; MAKE SURE UT
is IN HOURS

Delta = longitude;

theta = thetaG + Delta;

while theta > 360
    theta = theta - 360;
end % loop

LST = theta;

end % function
```

Published with MATLAB® R2023b

```
function nu = checkeclipse(R_sc,R_sun,radiusBody)
%{
Author: Justin Self
November 19, 2023
This function (straight from Curtis p.526; Algorithm 10.3) takes:

INPUTS:
    radiusBody = radius of central body, km
    R_sc = orbital radius of spacecraft from central body, km (ECI, earth)
    R_sun = sun vector from central body frame (ECI for earth) km

OUTPUT:
    nu = 0 == s/c is in shadow
    nu = 1 == s/c is NOT in shadow
%}
re = radiusBody;
r = norm(R_sc);
rsun = norm(R_sun);

theta = acos( dot(R_sun,R_sc)/(rsun*r) );
theta1 = acos(re/r);
theta2 = acos(re/rsun);

if theta1 + theta2 <= theta
    nu = 0;
    %disp("Nu is: " + nu + "; s/c is in SHADOW")
else
    nu = 1;
    %disp("Nu is: " + nu + "; s/c NOT in shadow")
end
```

Published with MATLAB® R2023b

```
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function [lamda eps r_S] = solar_position(jd)
%
% This function calculates the geocentric equatorial position vector
% of the sun, given the julian date.
%
% User M-functions required: None
% ------------------------------------------------------------------------
%...Astronomical unit (km):
AU    = 149597870.691;

%...Julian days since J2000:
n     = jd - 2451545;

%...Julian centuries since J2000:
cy    = n/36525;

%...Mean anomaly (deg{:
M     = 357.528 + 0.9856003*n;
M     = mod(M,360);

%...Mean longitude (deg):
L     = 280.460 + 0.98564736*n;
L     = mod(L,360);

%...Apparent ecliptic longitude (deg):
lamda = L + 1.915*sind(M) + 0.020*sind(2*M);
lamda = mod(lamda,360);

%...Obliquity of the ecliptic (deg):
eps   = 23.439 - 0.0000004*n;

%...Unit vector from earth to sun:
u     = [cosd(lamda); sind(lamda)*cosd(eps); sind(lamda)*sind(eps)];

%...Distance from earth to sun (km):
rS    = (1.00014 - 0.01671*cosd(M) - 0.000140*cosd(2*M))*AU;

%...Geocentric position vector (km):
r_S   = rS*u;
end %solar_position
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Published with MATLAB® R2023b

```
function aer = eci2aer(POSITION,UTC,LLA0,varargin)

%   ECI2AER Convert Earth-centered Inertial (ECI) to local azimuth,
%   elevation, and slant range coordinates.
%   AER =
ECI2AER( POSITION,UTC,LLA0,REDUCTION,DELTAAT,DELTAUT1,POLARMOTION,'ADDPARAMNA
ME',ADDPARAMVALUE )
%   converts a set of ECI Cartesian coordinates, POSITION, to local azimuth,
%   elevation and slant range coordinates.
%
%   Inputs arguments for ECI2AER are:
%   POSITION:       M-by-3 array of ECI coordinates in meters.
%   UTC:            Array of Universal Coordinated Time (UTC) in year,
%                   month, day, hour, minutes, and seconds for which the
%                   function calculates the coordinate conversion. Define
%                   the array as one of the following: Array with 1 row and
%                   6 columns, or M-by-6 array for M transformation
%                   matrices, one for each UTC date. Values for year,
%                   month, day, hour, and minutes should be whole numbers.
%   LLA0:           M-by-3 array with the geodetic coordinates of the local
%                   reference with latitude, longitude and ellipsoidal
%                   altitude in degrees, degrees and meters respectively.
%   REDUCTION:      String indicating the reduction process through which
%                   the function calculates the coordinate conversion.
%                   It can either be IAU-76/FK5 (which uses the IAU 1976
%                   Precession Model and the IAU 1980 Theory of Nutation,
%                   which is no longer current but some programs still use
%                   this reduction) or IAU-2000/2006 (which uses the P03
%                   precession model). The reduction method you select
%                   determines the ADDPARAMNAME parameter pair
%                   characteristics. The IAU-76/FK5 method returns a
%                   coordinate conversion that is not orthogonal due to the
%                   polar motion approximation. The default value is
%                   IAU-2000/2006.
%   DELTAAT:        Difference, in seconds, between the International Atomic
%                   Time (TAI) and UTC. It can be defined as either a
%                   scalar or a one-dimensional array with M elements (if M
%                   UTC dates are defined), for equal number ECI
%                   coordinates. The default value is an M-by-1 null array.
%   DELTAUT1:       Difference, in seconds, between UTC and Universal Time
%                   (UT1). It can be defined as either a scalar or a one-
%                   dimensional array with M elements (if M UTC dates
%                   defined) for equal number ECI coordinates. The default
%                   value is an M-by-1 null array.
%   POLARMOTION:    Polar displacement due to the motion of the Earth's
%                   crust, in radians, along the x- and y-axis. It can be
%                   defined as either a 1-by-2 array or an M-by-2 (if M UTC
%                   dates defined) for equal number of ECI coordinates. The
%                   default value is an M-by-2 null array.
%
%   Input parameter Name/Value pairs for 'ADDPARAMNAME' and ADDPARAMVALUE
%   are:
%   'DNUTATION':    (IAU-76/FK5 reduction only) M-by-2 array for the
```

```
%                   adjustment in radians to the longitude (dDeltaPsi) and
%                   obliquity (dDeltaEpsilon). The default value is an
%                   M-by-2 null array.
%   'DCIP':         (IAU-2000/2006 reduction only) M by 2 array for the
%                   adjustment in radians to the location of the Celestial
%                   Intermediate Pole (CIP) along the x (dDeltaX) and y
%                   (dDeltaY) axis. The default value is an M-by-2 null
%                   array.
%   'FLATTENING':   Earth's flattening. The default value is the one
%                   defined by WGS84 (1/298.257223563).
%   'RE':           Earth's equatorial radius. The default value is the one
%                   defined by WGS84 (6378137 meters).
%   'PSI0':         Angular direction of the local reference system (degrees
%                   clockwise from north). The default value is an M-by-1
%                   null array.
%
%   For historical values for DNUTATION and DCIP, see the International
%   Earth Rotation and Reference Systems Service (IERS) website
%   (http://www.iers.org) under the 'Earth Orientation Data' product.
%
%   Output calculated by ECI2AER is:
%   AER:    M-by-3 array with the local reference coordinates azimuth,
%           elevation, and slant range in degrees, degrees, and meters,
%           respectively. Azimuth is defined as the angle measured
%           clockwise from true north and varies between 0 and 360 deg.
%           Elevation is defined as the angle between a plane perpendicular
%           to the ellipsoid's surface at LL0 and the line that goes from
%           the local reference to the object's position and varies between
%           -90 and 90 degrees. Slant range is defined as the straight line
%           distance between the local reference and the object.
%
%   This method has higher accuracy over small distances from the local
%   geodetic reference frame (LLA0) in latitude and longitude, and nearer to
%   the equator.
%
%   Example:
%   Estimate the position of the azimuth, elevation and range for an object
%   with Earth Centered Inertial position 1e08*[-3.8454 -0.5099 -0.3255]
%   meters for the date 1969/7/20 21:17:40 UTC at 28.4 deg North, 80.5 deg
%   West and 2.7 meters altitude.
%
%   AER = eci2aer(1e08*[-3.8454,-0.5099,-0.3255],[1969,7,20,21,17,40], ...
%                 [28.4,-80.5,2.7])
%
%   See also LLA2ECI, ECI2LLA, DCMECI2ECEF, FLAT2LLA, LLA2FLAT, DELTAUT1,
%   DELTACIP, POLARMOTION.
```

Validate i/o

```
% Validate outputs
nargoutchk(0,1)

% Validate date
```

```
validateattributes(UTC,{'numeric'},{'ncols',6,'real','finite','nonnan'})

% Validate latitude longitude altitude
validateattributes(POSITION,{'numeric'},{'ncols',3,'real','finite','nonnan'})

% Assign date vectors
year = UTC(:,1);
month = UTC(:,2);
day = UTC(:,3);
hour = UTC(:,4);
min = UTC(:,5);
sec = UTC(:,6);

% Validate vectors
if any(year<1)
    error(message('aero:dcmeci2ecef:invalidYear'));
end
if any(month<1) || any(month>12)
    error(message('aero:dcmeci2ecef:invalidMonth'));
end
if any(day<1) || any(day>31)
    error(message('aero:dcmeci2ecef:invalidDay'));
end
if any(hour<0) || any(hour>24)
    error(message('aero:dcmeci2ecef:invalidHour'));
end
if any(min<0) || any(min>60)
    error(message('aero:dcmeci2ecef:invalidMin'));
end
if any(sec<0) || any(sec>60)
    error(message('aero:dcmeci2ecef:invalidSec'));
end
len = length(year);

% Parse and validate the inputs
ob = inputParser;
validReduction = {'IAU-2000/2006','IAU-76/FK5'};
addRequired(ob,'POSITION',@(x) validateattributes(x,{'numeric'},
{'ncols',3,'real',...
    'finite','nonnan','size',[len,3]}));
addRequired(ob,'UTC',@(x) validateattributes(x,{'numeric'},
{'ncols',6,'real',...
    'finite','nonnan'}));
addRequired(ob,'LLA0',@(x) validateattributes(x,{'numeric'},
{'ncols',3,'real',...
    'finite','nonnan','size',[len,3]}));
addOptional(ob,'reduction','iau-2000/2006',@(x)
validateReduction(x,validReduction));
addOptional(ob,'deltaAT',zeros(len,1),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,1]}));
addOptional(ob,'deltaUT1',zeros(len,1),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,1]}));
```

```
addOptional(ob,'polarMotion',zeros(len,2),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,2]}));
addParameter(ob,'dNutation',zeros(len,2),@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[len,2]}));
addParameter(ob,'dCIP',zeros(len,2),@(x) validateattributes(x,{'numeric'},...
    {'real','finite','nonnan','size',[len,2]}));
addParameter(ob,'flattening',1/298.257223563,@(x) validateattributes(x,
{'numeric'},...
    {'real','finite','nonnan','size',[1 1]}));
addParameter(ob,'RE',6378137,@(x) validateattributes(x,{'numeric'},...
    {'real','finite','nonnan','size',[1 1]}));
addOptional(ob,'PSI0',zeros(len,1),@(x) validateattributes(x,{'numeric'},...
    {'real','finite','nonnan','size',[len,1]}));

% Parse input object
parse(ob,POSITION,UTC,LLA0,varargin{:});

% Validate reduction
reduction = ob.Results.reduction;
reduction = lower(validatestring(reduction,validReduction));

%Validate that the additional parameter matches the reduction method
if any(strcmp(ob.UsingDefaults,'dNutation')) &&
~any(strcmp(ob.UsingDefaults,'dCIP')) &&...
        ~strcmp(reduction,'iau-2000/2006')
    error(message('aero:dcmeci2ecef:invalidDNutation'));
elseif ~any(strcmp(ob.UsingDefaults,'dNutation')) &&
any(strcmp(ob.UsingDefaults,'dCIP')) && ...
        ~strcmp(reduction,'iau-76/fk5')
    error(message('aero:dcmeci2ecef:invalidDCIP'));
end

%Validate that both reduction methods are not defined (just one should be
%defined)
if ~any(strcmp(ob.UsingDefaults,'dNutation')) &&
~any(strcmp(ob.UsingDefaults,'dCIP'))
    error(message('aero:dcmeci2ecef:invalidDNutationDCIP'))
end
```

Calculate DCM ECI to ECEF

```
switch reduction
    case 'iau-76/fk5'
        dcm =
dcmeci2ecef(ob.Results.reduction,ob.Results.UTC,ob.Results.deltaAT,...

ob.Results.deltaUT1,ob.Results.polarMotion,'dNutation',ob.Results.dNutation);
    case 'iau-2000/2006'
        dcm =
dcmeci2ecef(ob.Results.reduction,ob.Results.UTC,ob.Results.deltaAT,...
```

```
ob.Results.deltaUT1,ob.Results.polarMotion,'dCIP',ob.Results.dCIP);
end
```

Calculate position in ECEF coordinates

```
tmp = arrayfun(@(k) (dcm(:,:,k)*POSITION(k,:)'),1:len,'UniformOutput',false);
ecefObject = cell2mat(tmp)';
```

Calculate position in NED coordinates

Wrap latitude and longitude for the local geodetic reference system

```
LLA0 = ob.Results.LLA0;
[~, LLA0(:,1), LLA0(:,2)] = wraplatitude( ob.Results.LLA0(:,1),
ob.Results.LLA0(:,2),180);
[~, LLA0(:,2)] = wraplongitude( LLA0(:,2), 180 );
% Use local function for NED transform
tmp2 = arrayfun(@(k)
(nedCalc(ecefObject(k,:),LLA0(k,:),ob.Results.flattening,ob.Results.RE)),1:le
n,'UniformOutput',false);
nedObject = cell2mat(tmp2');
```

Calculate Azimuth, Elevation, Range

```
hypotxy = hypot(nedObject(:,1),nedObject(:,2));
r = hypot(hypotxy,-nedObject(:,3));
elev = atan2d(-nedObject(:,3),hypotxy);
az = atan2d(nedObject(:,2),nedObject(:,1))-ob.Results.PSI0;
az = mod(az,360);
aer = [az elev r];

end

function validateReduction(reduction,validReduction)
validatestring(reduction,validReduction);
end

function ned = nedCalc(posEcef,LLA0,f,Re)
% This helper function calculates the NED coordinates for the object given
% in ECEF coordinates given a local geodetic position LLA0. It requires the
% flattening and equatorial radius. It calculates the ENU coordinates and
% finally rotates it for NED.

refPosEcef = lla2ecef(LLA0,f,Re);
dPos = posEcef - refPosEcef;

% ENU position
enu = [ -sind(LLA0(2))                  cosd(LLA0(2))                    0;...
        -cosd(LLA0(2))*sind(LLA0(1))    -sind(LLA0(1))*sind(LLA0(2))
cosd(LLA0(1)); ...
        cosd(LLA0(1))*cosd(LLA0(2))     cosd(LLA0(1))*sind(LLA0(2))
sind(LLA0(1))]*...
```

```
        dPos';

% NED position
ned = [enu(2) enu(1) -enu(3)];

end
```


Published with MATLAB® R2023b

```
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function [lamda eps r_S] = solar_position(jd)
%
% This function calculates the geocentric equatorial position vector
% of the sun, given the julian date.
%
% User M-functions required: None
% ------------------------------------------------------------------------
%...Astronomical unit (km):
AU    = 149597870.691;

%...Julian days since J2000:
n     = jd - 2451545;

%...Julian centuries since J2000:
cy    = n/36525;

%...Mean anomaly (deg{:
M     = 357.528 + 0.9856003*n;
M     = mod(M,360);

%...Mean longitude (deg):
L     = 280.460 + 0.98564736*n;
L     = mod(L,360);

%...Apparent ecliptic longitude (deg):
lamda = L + 1.915*sind(M) + 0.020*sind(2*M);
lamda = mod(lamda,360);

%...Obliquity of the ecliptic (deg):
eps   = 23.439 - 0.0000004*n;

%...Unit vector from earth to sun:
u     = [cosd(lamda); sind(lamda)*cosd(eps); sind(lamda)*sind(eps)];

%...Distance from earth to sun (km):
rS    = (1.00014 - 0.01671*cosd(M) - 0.000140*cosd(2*M))*AU;

%...Geocentric position vector (km):
r_S   = rS*u;
end %solar_position
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Published with MATLAB® R2023b

```
function [r,v] = universalVar(mu,deltat,r0Vect,v0Vect,TOL)

% Self, Justin
% Aero351: Orbital Mechanics I

% This function uses Newton's method to solve the Stumpf functions for the
Universal Variable

% Inputs:
% mu: gravitational parameter of planet of interest
% deltat: time interval (fixed) desired between iterations
% v0Vect = initial velocity vector, 3x1
% r0Vect = initial position vector, 3x1
% TOL = tolerance desired (error); use 1e-8

% Output:
% [r, v] = new r, v vectors

% UNIVERSAL VARIABLE ALGORITHM

% ALGORITHM 3.4, Part 1.
% i. Find r and v
r0 = norm(r0Vect);
v0 = norm(v0Vect);

% ii. Find vr|0 by projecting v0 onto onto the direction of r0
vr0 = dot(v0Vect,r0Vect)* r0^(-1);

% iii. The reciprical of the semimajor axis, alpha, (using Eqn. 3.48)
alpha = (2/r0)  - (v0^2 / mu);

% OPTIONAL:
%{
% Learn something from the sign of alpha
if alpha < 0
    disp("This is a hyperbolic trajectory")
elseif alpha > 0
    disp("This is an elliptical orbit")
else
    disp("This is a parabolic orbit.")
end
%}

% ALGORITHM 3.4, Part 2.

% Use Algorithm 3.3 to find the universal anomaly X.
% i. use Eq. 3.66 for an initial estimate of X0.
% ii - v, use homebuilt Universal Variable function to solve for X.
```

Solve for X (UniversalVariable)

Figure out the best initial guess

```
x0 = sqrt(mu)*abs(alpha)*deltat;

%z = alpha*(X^2);

% Define f and fprime functions
f = @(X) ((r0 * vr0) / (mu^.5)) * (X^2) * Cstumpf(alpha,X) + (1 -
(alpha*r0))*(X^3) * Sstumpf(alpha,X) + r0*X - ((mu^.5)*deltat);
fprime = @(X) ((r0 * vr0) / (mu^.5)) * X * (1 -
alpha*(X^2)*Sstumpf(alpha,X))  + (1 - (alpha*r0)) *(X^2)* Cstumpf(alpha,X) +
r0;

% Newton's Method
x1 = x0 - f(x0) * (fprime(x0))^-1;
x = [x0; x1];
err = abs(x1 - x0);

while err > TOL
    x0 = x1;
    x1 = x0 - f(x0) *(fprime(x0))^-1;
    err = abs(x1 - x0);
    x = [x; x1]; % fix this??
end

X = x(end);
```

Use X to find new r, v vectors

```
% ALGORITHM 3.4, Part 3.
% Use Eqns (3.69a) and (3.69b) to obtain f and g
f = 1 - (X^2 / r0) * Cstumpf(alpha,X);
g = deltat - mu^(-.5)*(X^3)* Sstumpf(alpha,X);

% ALGORITHM 3.4, Part 4.
% Use Eqn (3.67) to compute r_vect_f and r_mag_f
rf_vect = f*r0Vect + g*v0Vect;
rfmag = norm(rf_vect);

% ALGORITHM 3.4, Part 5.
% Use Eqns (3.69c) and (3.69d) to obtain fdot and gdot.
fdot = ((mu^.5) / (rfmag*r0)) * (alpha*(X^3)*Sstumpf(alpha,X) - X);
gdot = 1 - ((X^2) / rfmag)*Cstumpf(alpha,X);

% ALGORITHM 3.4, Part 6.
vf_vect = fdot*r0Vect + gdot*v0Vect;
vfmag = norm(vf_vect);

% FINAL RESULTS
r = rf_vect;
v = vf_vect;

end % function
```

Published with MATLAB® R2023b

```
function [r1,r2,r3,v2,rhohatVect,tau1,tau3,RsVect] = IOD_Gauss(lat, lon,
altSite, RA, dec, time, extended)

%{
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Winter 2024: AERO 557 | Advanced Orbital Mechanics
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
This function implements the Gauss method for initial orbit determination
(IOD) with the option to perform the IOD with NONEXTENDED (class
only) or EXTENDED (traditional) methods.

This method performs "remarkably well when the data is separated by 10
degrees or less... five to ten minutes apart in low-Earth satellites."
(Vallado 2007). This method assumes all three observations lie in the same
plane.

Frame: topocentric (local)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
INPUTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        lat     = latitude of site [degrees]
        lon     = longitude of site [degrees]
        altSite = altitude of site [m]
        RA      = 3x1 vector: Right Ascension [degrees]
        dec     = 3x1 vector: Declination [degrees]
        time    = 3x6 vector containing UTC times:

            % Example:
            t1 = datetime('2013-03-25  03:10:30.032','InputFormat','yyyy-MM-
dd HH:mm:ss.SSS');
            t2 = datetime('2013-03-25  03:15:20.612','InputFormat','yyyy-MM-
dd HH:mm:ss.SSS');
            t3 = datetime('2013-03-25  03:20:32.777','InputFormat','yyyy-MM-
dd HH:mm:ss.SSS');

            time = [datevec(t1);datevec(t2);datevec(t3)];

        extended (OPTIONAL) = leave blank or enter 'yes' or 'Yes' for
                                extended version
                            = 'no' or (anything else) for NON extended
                            version

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
OUTPUTS:   (in ECI)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        r(i)    = ECI orbital radius per observation [km]
        v(i)    = ECI orbital velocity               [km/s]
%}

% Begin function

    % Constants
```

```
    Re = 6378; % radius of earth, equator; km
    Rp = 6357; % radius of earth, polar; km
    mu = 398600; % km3/s2

    % Input time vectors for each observation
    t1 = time(1,:);
    t2 = time(2,:);
    t3 = time(3,:);

    %.... Convert time vectors to julian date
    [jd1, ~, ~] = juliandateFunction(t1(1),t1(2),t1(3),[t1(4) t1(5) t1(6)]);
    [jd2, ~, ~] = juliandateFunction(t2(1),t2(2),t2(3),[t2(4) t2(5) t2(6)]);
    [jd3, ~, ~] = juliandateFunction(t3(1),t3(2),t3(3),[t3(4) t3(5) t3(6)]);

    % Input right ascension and dec for each observation
    ra1 = RA(1); % deg
    ra2 = RA(2); % deg
    ra3 = RA(3); % deg

    dec1 = dec(1); % deg
    dec2 = dec(2); % deg
    dec3 = dec(3); % deg
```

Gauss NON EXTENDED case

```
        %.... Define variables used in Vallado derivation (ALGORITHM 51)
        tau1 = (jd1 - jd2); % in days
        tau3 = (jd3 - jd2); % in days

        a1 = tau3 / (tau3 - tau1);
        a1u = 86400*86400*(tau3 * ((tau3 - tau1)^2  - tau3^2)) / (6 * (tau3
- tau1) );
        a3 = -tau1 / (tau3 - tau1);
        a3u = (86400*86400)*-(tau1 * ((tau3 - tau1)^2  - tau1^2)) / (6 *
(tau3 - tau1) );

        %.... Define L matrix (see Vallado page 30)
        L1 = [cosd(dec1)*cosd(ra1); cosd(dec1)*sind(ra1); sind(dec1)];
        L2 = [cosd(dec2)*cosd(ra2); cosd(dec2)*sind(ra2); sind(dec2)];
        L3 = [cosd(dec3)*cosd(ra3); cosd(dec3)*sind(ra3); sind(dec3)];

        L = [L1,L2,L3];

        % Find L^-1
        %Linv_vallado = det(L)^-1 * [    L2(2)*L3(3) - L3(2)*L2(3),
-L1(2)*L3(3) + L3(2)*L1(3),  L1(2)*L2(3) - L2(2)*L1(3);
                                %  -L2(1)*L3(3) + L3(1)*L2(3),  L1(1)*L3(3) -
L3(1)*L1(3), -L1(1)*L2(3) + L2(1)*L1(3);
                                 % L2(1)*L3(2) - L3(1)*L2(2),  -L1(1)*L3(2) +
L3(1)*L1(2),  L1(1)*L2(2) - L2(1)*L1(2)];
        Linv_matlab = inv(L); % they are not the same!! Suspect error in
Vallado ^
```

```
        %.... Compute Rsite in ECI
        % Set up ECI frame
        Ihat = [1 0 0];
        Jhat = [0 1 0];
        Khat = [0 0 1];

        % Compute LST for each observation
        theta1 = local_sidereal_time_justin(t1(1),t1(2),t1(3),[t1(4) t1(5)
t1(6)],lon);
        theta2 = local_sidereal_time_justin(t2(1),t2(2),t2(3),[t2(4) t2(5)
t2(6)],lon);
        theta3 = local_sidereal_time_justin(t3(1),t3(2),t3(3),[t3(4) t3(5)
t3(6)],lon);
        theta = [theta1;theta2;theta3];

        % f == oblateness factor
        f = (Re - Rp) / (Re);
        H = altSite/1000; % altitude of site, CONVERTED meters to KM

        % in ECI
        RSITE = zeros(3,3); % preallocate
        K_term = ( (Re * (1-f)^2 ) / (sqrt(1 - (2*f - f^2)*sind(lat)^2)  ) +
H)*sind(lat)*Khat;

        %.... Find Rsite in ECI
        for i = 1:3
            RSITE(:,i) = ((Re / (sqrt(1 - (2*f - f^2) * sind(lat)^2 ) ) ) +
H) * cosd(lat) * (cosd(theta(i)) * Ihat + sind(theta(i)) * Jhat) + K_term;
        end

        %..... Define M
        M = Linv_matlab*RSITE;

        %..... Define rho2 in terms of d1 and d2
        d1 = M(2,1)*a1 - M(2,2) + M(2,3)*a3;
        d2 = M(2,1)*a1u + M(2,3)*a3u;

        %..... Define C, the dot product, to get an 8th deg poly:
        Rs1 = [RSITE(1,1); RSITE(2,1); RSITE(3,1)];
        Rs2 = [RSITE(1,2); RSITE(2,2); RSITE(3,2)]; % second R_site vector
(3x1)
        Rs3 = [RSITE(1,3); RSITE(2,3); RSITE(3,3)];
        C = dot(L2,Rs2);

        %..... Find the roots of the eigth-degree polynomial
        % 0 = r2^8 - (d1^2 + 2*C*d1 + rsite2^2)*r2^6 - 2*mu*(C*d2 +
        % d1*d2)*r2^3 - mu^2*d2^2
        rsite2 = norm(Rs2);
        cf8 = 1;
        cf7 = 0;
        cf6 = -(d1^2 + 2*C*d1 + rsite2^2);
        cf5 = 0;
        cf4 = 0;
        cf3 = -2*mu*(C*d2 + d1*d2);
```

```
        cf2 = 0;
        cf1 = 0;
        cf0 = -mu^2*d2^2;
        p = [cf8 cf7 cf6 cf5 cf4 cf3 cf2 cf1 cf0]; % coefficients in
polynomial
        r = roots(p);

        % Find REAL and POSITIVE root (only one)
        TOL = 1e-10; % tolerance for calling a 0ish imaginary part REAL
        index = abs(imag(r)) < TOL; % absolute val of imaginary check of
roots vector
        reals = r(index); % real part(s)
        positive = reals > 0; % extract only POSITIVE value(s)

        % the root!
        r2 = reals(positive > 0);

        %.... Compute slant ranges!
        u = mu / r2^3;
        c1 = a1 + a1u * u;
        c2 = -1; % part of our assumptions starting out
        c3 = a3 + a3u*u;

        %.... Gauss method initial guesses
        rho1 = (-c1*M(1,1) - c2*M(1,2) - c3*M(1,3) ) / c1;
        rho2 = (-c1*M(2,1) - c2*M(2,2) - c3*M(2,3) ) / c2;
        rho3 = (-c1*M(3,1) - c2*M(3,2) - c3*M(3,3) ) / c3;

        %.... Compute final r_vector

        % Final results for non-extended case
        % (Gauss method formally ends here).
        r1 = norm(rho1)*L1 + Rs1;
        r2 = norm(rho2)*L2 + Rs2;
        r3 = norm(rho3)*L3 + Rs3;

        % Vallado: "Use Gibbs to find v2"
        [v2_gibbs, ~,~,~, ~] = gibbs(r1,r2,r3);
        %[v2_hgibbs, ~,~,~, ~ ] = hgibbs(r1,r2,r3,jd1,jd2,jd3 );

        % choose one for output
        v2 = v2_gibbs;

        % Additional outputs (useful for Double-R inputs)
        % rhohat is useful for Gauss Extended
        rhohat1 = [cosd(dec1)*cosd(ra1); cosd(dec1)*sind(ra1); sind(dec1)];
        rhohat2 = [cosd(dec2)*cosd(ra2); cosd(dec2)*sind(ra2); sind(dec2)];
        rhohat3 = [cosd(dec3)*cosd(ra3); cosd(dec3)*sind(ra3); sind(dec3)];
        rhohatVect = [rhohat1,rhohat2,rhohat3];
        RsVect = [Rs1,Rs2,Rs3];

    % end of Gauss (nonextended) method
```

Gauss Extended

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Check if user wants "extended" Gauss version or not
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % defaults "extended" to 'yes' if no input
    if (nargin < 7) || isempty(extended)
        extended = 'yes';
    end

    % Take user input for EXTENDED or NON-EXTENDED cases
    if strcmpi(extended,'yes')

        % Gauss extended algorithm goes here.
        % Convert tau (days) to seconds (required in functs)

        tau1 = tau1*86400;
        tau3 = tau3*86400;

        f1 = 1 - u/2 * tau1^2;
        g1 = tau1 - (1/6) * (mu/norm(r2)^3) * tau1^3;
        f3 = 1 - u/2 * tau3^2;
        g3 = tau3 - (1/6) * (mu/norm(r2)^3) * tau3^3;

      % Call Vallado
      [r1, r2, r3, v2] =
AERO557GaussExtensionVallado(r2,v2,rho1,rho2,rho3,f1,g1,f3,g3,rhohat1,rhohat2
,rhohat3,mu,M,tau1,tau3,Rs1,Rs2,Rs3);

    end % gauss extended

end  % function
```

Published with MATLAB® R2023b

Double R

```
%-------------------------------------------------------------------------
    function [r2vec, v2vec] = AERO557doubleR(L1hat,L2hat,L3hat, R1,R2, R3,
tau1,tau3)

        %initial guess for the iteration
        r1 = 2*6378;
        r2 = 2.01*6378;
        c1 = dot(2*L1hat,R1);
        c2 = dot(2*L2hat,R2);
        count = 1;
        Err = 1;
        while Err > .001

[F1,F2,f,g,r3vec,r2vec]=AERO557drfunc(c1,c2,r1,r2,tau1,tau3,L1hat,L2hat,L3hat
,R1,R2,R3);
            dr1 = .005*r1;
            dr2 = .005*r2;

[F1r1dr1,F2r1dr1,~,~]=AERO557drfunc(c1,c2,r1+dr1,r2,tau1,tau3,L1hat,L2hat,L3h
at,R1,R2,R3);

[F1r2dr2,F2r2dr2,~,~]=AERO557drfunc(c1,c2,r1,r2+dr2,tau1,tau3,L1hat,L2hat,L3h
at,R1,R2,R3);

            dF1dr1 = (F1r1dr1 - F1)/dr1;
            dF2dr1 = (F2r1dr1 - F2)/dr1;
            dF1dr2 = (F1r2dr2 - F1)/dr2;
            dF2dr2 = (F2r2dr2 - F2)/dr2;

            %run the function with the dr conditions

            delta = dF1dr1*dF2dr2 - dF2dr1*dF1dr2;
            delta1 = dF2dr2*F1 - dF1dr2*F2;
            delta2 = dF1dr1*F2 - dF2dr1*F1;
            dr1 = -delta1/delta;
            dr2 = -delta2/delta;

            Err = (abs(dr1) + abs(dr2))/2;
            r1 = r1 + dr1;
            r2 = r2 + dr2;
            count=count+1;

        end

        %after convergence
        v2vec = (r3vec - f*r2vec)/g;
        % no of interations to converge
        disp("Number of iterations for Double R to converge is: " + count)
```

double R function

```
    function
[F1,F2,f,g,r3vec,r2vec]=AERO557drfunc(c1,c2,r1,r2,tau1,tau3,L1hat,L2hat,L3hat
,R1,R2,R3)
        q1 = (-c1+sqrt(c1^2 - 4*(dot(R1,R1)-r1^2)))/2;
        q2 = (-c2+sqrt(c2^2 - 4*(dot(R2,R2)-r2^2)))/2;

        r1vec = R1 + q1*L1hat;
        r1 = norm(r1vec);
        r2vec = R2 + q2*L2hat;
        r2 = norm(r2vec);

        what = cross(r1vec,r2vec)/(norm(r1vec)*norm(r2vec));
        q3 = dot(-R3,what)/dot(L3hat,what);
        r3vec = R3 + q3*L3hat;
        r3 = norm(r3vec);

        %difference angles
        costheta21 = dot(r2vec,r1vec)/(norm(r2vec)*norm(r1vec));
        t21 = acos(costheta21);
        costheta31 = dot(r3vec,r1vec)/(norm(r3vec)*norm(r1vec));
        t31 = acos(costheta31);
        costheta32 = dot(r3vec,r2vec)/(norm(r3vec)*norm(r2vec));
        t32 = acos(costheta32);

        % if what(3)>0
        sintheta21 = sqrt(1-costheta21^2);
        sintheta31 = sqrt(1-costheta31^2);
        sintheta32 = sqrt(1-costheta32^2);
        % elseif what(3)<0
        %     sintheta21 = -sqrt(1-costheta21^2);
        %     sintheta31 = -sqrt(1-costheta31^2);
        %     sintheta32 = -sqrt(1-costheta32^2);
        % end

        theta31 = acos(costheta31);
        if theta31>pi
            c1 = (r2*sintheta32)/(r1*sintheta31);
            c3 = (r2*sintheta21)/(norm(r3vec)*sintheta31);
            p = (c1*r1 + c3*r3 - r2)/(c1 + c3 -1);
        elseif theta31<=pi
            c1 = (r1*sintheta31)/(r2*sintheta32);
            c3 = (r1*sintheta21)/(r3*sintheta32);
            p = (c3*r3 - c1*r2 + r1)/(-c1 + c3 +1);
        end

        ecostheta1 = p/r1 -1;
        ecostheta2 = p/r2 -1;
        ecostheta3 = p/r3 -1;

        theta21 = acos(costheta21);
```

```
        if theta21 ~= pi
            esintheta2 = (-costheta21*ecostheta2 + ecostheta1)/sintheta21;
        else
            esintheta2 = (costheta32*ecostheta2 - ecostheta3)/sintheta31;
        end

        e2 = ecostheta2^2 + esintheta2^2;
        a = p/(1-e2);

        if sqrt(e2)<1
            n = sqrt(398600/a^3);
            S = r2/p*sqrt(1-e2)*esintheta2;
            C = r2/p*(e2 + ecostheta2);
            sinE32 = r3/sqrt(a*p)*sintheta32 - r3/p*(1 - costheta32)*S;
            cosE32 = 1 - (r2*r3)/(a*p)*(1-costheta32);
            sinE21 = r1/sqrt(a*p)*sintheta21 + r1/p*(1 - costheta21)*S;
            cosE21 = 1 - (r2*r1)/(a*p)*(1-costheta21);
            M32 = acos(cosE32) + 2*S*sin(acos(cosE32)/2)^2 - C*sinE32;
            M12 = -acos(cosE21) + 2*S*sin(acos(cosE21)/2)^2 + C*sinE21;
            F1 = tau1 - M12/n;
            F2 = tau3 - M32/n;
            f = 1 - a/r2*(1 - cosE32);
            g = tau3 - sqrt(a^3/398600)*(acos(cosE32) - sinE32);
        else
            n = sqrt(398600/-a^3);
            Sh = r2/p*sqrt(e2-1)*esintheta2;
            Ch = r2/p*(e2 + ecostheta2);
            sinhF32 = r3/sqrt(-a*p)*sintheta32 - r3/p*(1 - costheta32)*Sh;
            F32 = log(sinhF32 + sqrt(sinhF32 + 1));
            sinhF21 = r1/sqrt(-a*p)*sintheta21 + r1/p*(1 - costheta32)*Sh;
            F21 = log(sinhF21 + sqrt(sinhF21^2 + 1));
            M32 = -F32 + 2*Sh*sinh(F32/2)^2 + Ch*sinhF32;
            M12 = F21 + 2*Sh*sinh(F21/2)^2 + Ch*sinhF21;
            F1 = tau1 - M12/n;
            F2 = tau3 - M32/n;
            f = 1 - (-a)/r2*(1 - cosh(F32));
            g = tau3 - sqrt((-a)^3/398600)*(F32 - sinhF32);
        end

    end %drfunc

    end %main
```

Published with MATLAB® R2023b

Double R

```
%-------------------------------------------------------------------------
    function [r2vec, v2vec] = AERO557doubleR(L1hat,L2hat,L3hat, R1,R2, R3,
tau1,tau3)

        %initial guess for the iteration
        r1 = 2*6378;
        r2 = 2.01*6378;
        c1 = dot(2*L1hat,R1);
        c2 = dot(2*L2hat,R2);
        count = 1;
        Err = 1;
        while Err > .001

[F1,F2,f,g,r3vec,r2vec]=AERO557drfunc(c1,c2,r1,r2,tau1,tau3,L1hat,L2hat,L3hat
,R1,R2,R3);
            dr1 = .005*r1;
            dr2 = .005*r2;

[F1r1dr1,F2r1dr1,~,~]=AERO557drfunc(c1,c2,r1+dr1,r2,tau1,tau3,L1hat,L2hat,L3h
at,R1,R2,R3);

[F1r2dr2,F2r2dr2,~,~]=AERO557drfunc(c1,c2,r1,r2+dr2,tau1,tau3,L1hat,L2hat,L3h
at,R1,R2,R3);

            dF1dr1 = (F1r1dr1 - F1)/dr1;
            dF2dr1 = (F2r1dr1 - F2)/dr1;
            dF1dr2 = (F1r2dr2 - F1)/dr2;
            dF2dr2 = (F2r2dr2 - F2)/dr2;

            %run the function with the dr conditions

            delta = dF1dr1*dF2dr2 - dF2dr1*dF1dr2;
            delta1 = dF2dr2*F1 - dF1dr2*F2;
            delta2 = dF1dr1*F2 - dF2dr1*F1;
            dr1 = -delta1/delta;
            dr2 = -delta2/delta;

            Err = (abs(dr1) + abs(dr2))/2;
            r1 = r1 + dr1;
            r2 = r2 + dr2;
            count=count+1;

        end

        %after convergence
        v2vec = (r3vec - f*r2vec)/g;
        % no of interations to converge
        disp("Number of iterations for Double R to converge is: " + count)
```

double R function

```
    function
[F1,F2,f,g,r3vec,r2vec]=AERO557drfunc(c1,c2,r1,r2,tau1,tau3,L1hat,L2hat,L3hat
,R1,R2,R3)
        q1 = (-c1+sqrt(c1^2 - 4*(dot(R1,R1)-r1^2)))/2;
        q2 = (-c2+sqrt(c2^2 - 4*(dot(R2,R2)-r2^2)))/2;

        r1vec = R1 + q1*L1hat;
        r1 = norm(r1vec);
        r2vec = R2 + q2*L2hat;
        r2 = norm(r2vec);

        what = cross(r1vec,r2vec)/(norm(r1vec)*norm(r2vec));
        q3 = dot(-R3,what)/dot(L3hat,what);
        r3vec = R3 + q3*L3hat;
        r3 = norm(r3vec);

        %difference angles
        costheta21 = dot(r2vec,r1vec)/(norm(r2vec)*norm(r1vec));
        t21 = acos(costheta21);
        costheta31 = dot(r3vec,r1vec)/(norm(r3vec)*norm(r1vec));
        t31 = acos(costheta31);
        costheta32 = dot(r3vec,r2vec)/(norm(r3vec)*norm(r2vec));
        t32 = acos(costheta32);

        % if what(3)>0
        sintheta21 = sqrt(1-costheta21^2);
        sintheta31 = sqrt(1-costheta31^2);
        sintheta32 = sqrt(1-costheta32^2);
        % elseif what(3)<0
        %     sintheta21 = -sqrt(1-costheta21^2);
        %     sintheta31 = -sqrt(1-costheta31^2);
        %     sintheta32 = -sqrt(1-costheta32^2);
        % end

        theta31 = acos(costheta31);
        if theta31>pi
            c1 = (r2*sintheta32)/(r1*sintheta31);
            c3 = (r2*sintheta21)/(norm(r3vec)*sintheta31);
            p = (c1*r1 + c3*r3 - r2)/(c1 + c3 -1);
        elseif theta31<=pi
            c1 = (r1*sintheta31)/(r2*sintheta32);
            c3 = (r1*sintheta21)/(r3*sintheta32);
            p = (c3*r3 - c1*r2 + r1)/(-c1 + c3 +1);
        end

        ecostheta1 = p/r1 -1;
        ecostheta2 = p/r2 -1;
        ecostheta3 = p/r3 -1;

        theta21 = acos(costheta21);
```

```
        if theta21 ~= pi
            esintheta2 = (-costheta21*ecostheta2 + ecostheta1)/sintheta21;
        else
            esintheta2 = (costheta32*ecostheta2 - ecostheta3)/sintheta31;
        end

        e2 = ecostheta2^2 + esintheta2^2;
        a = p/(1-e2);

        if sqrt(e2)<1
            n = sqrt(398600/a^3);
            S = r2/p*sqrt(1-e2)*esintheta2;
            C = r2/p*(e2 + ecostheta2);
            sinE32 = r3/sqrt(a*p)*sintheta32 - r3/p*(1 - costheta32)*S;
            cosE32 = 1 - (r2*r3)/(a*p)*(1-costheta32);
            sinE21 = r1/sqrt(a*p)*sintheta21 + r1/p*(1 - costheta21)*S;
            cosE21 = 1 - (r2*r1)/(a*p)*(1-costheta21);
            M32 = acos(cosE32) + 2*S*sin(acos(cosE32)/2)^2 - C*sinE32;
            M12 = -acos(cosE21) + 2*S*sin(acos(cosE21)/2)^2 + C*sinE21;
            F1 = tau1 - M12/n;
            F2 = tau3 - M32/n;
            f = 1 - a/r2*(1 - cosE32);
            g = tau3 - sqrt(a^3/398600)*(acos(cosE32) - sinE32);
        else
            n = sqrt(398600/-a^3);
            Sh = r2/p*sqrt(e2-1)*esintheta2;
            Ch = r2/p*(e2 + ecostheta2);
            sinhF32 = r3/sqrt(-a*p)*sintheta32 - r3/p*(1 - costheta32)*Sh;
            F32 = log(sinhF32 + sqrt(sinhF32 + 1));
            sinhF21 = r1/sqrt(-a*p)*sintheta21 + r1/p*(1 - costheta32)*Sh;
            F21 = log(sinhF21 + sqrt(sinhF21^2 + 1));
            M32 = -F32 + 2*Sh*sinh(F32/2)^2 + Ch*sinhF32;
            M12 = F21 + 2*Sh*sinh(F21/2)^2 + Ch*sinhF21;
            F1 = tau1 - M12/n;
            F2 = tau3 - M32/n;
            f = 1 - (-a)/r2*(1 - cosh(F32));
            g = tau3 - sqrt((-a)^3/398600)*(F32 - sinhF32);
        end

    end %drfunc

    end %main
```

Published with MATLAB® R2023b

```
% LAMBERT            Lambert-targeter for ballistic flights
%                    (Izzo, and Lancaster, Blanchard & Gooding)
%
% Usage:
%    [V1, V2, extremal_distances, exitflag] = lambert(r1, r2, tf, m,
GM_central)
%
% Dimensions:
%             r1, r2 ->  [1x3]
%             V1, V2 ->  [1x3]
% extremal_distances ->  [1x2]
%              tf, m ->  [1x1]
%         GM_central ->  [1x1]
%
% This function solves any Lambert problem *robustly*. It uses two separate
% solvers; the first one tried is a new and unpublished algorithm developed
% by Dr. D. Izzo from the European Space Agency [1]. This version is
extremely
% fast, but especially for larger [m] it still fails quite frequently. In
such
% cases, a MUCH more robust algorithm is started (the one by Lancaster &
% Blancard [2], with modifcations, initial values and other improvements by
% R.Gooding [3]), which is a lot slower partly because of its robustness.
%
% INPUT ARGUMENTS:
% ========================================================================
%    name        units    description
% ========================================================================
%   r1, r1       [km]     position vectors of the two terminal points.
%     tf        [days]    time of flight to solve for
%      m          [-]     specifies the number of complete orbits to complete
%                         (should be an integer)
% GM_central   [km3/s2]   std. grav. parameter (G•M = mu) of the central body
%
% OUTPUT ARGUMENTS:
% ========================================================================
%   name             units   description
% ========================================================================
%  V1, V2             [km/s]  terminal velocities at the end-points
%  extremal_distances  [km]   minimum(1) and maximum(2) distance of the
%                             spacecraft to the central body.
%  exitflag             [-]   Integer containing information on why the
%                             routine terminated. A value of +1 indicates
%                             success; a normal exit. A value of -1
%                             indicates that the given problem has no
%                             solution and cannot be solved. A value of -2
%                             indicates that both algorithms failed to find
%                             a solution. This should never occur since
%                             these problems are well-defined, and at the
%                             very least it can be determined that the
%                             problem has no solution. Nevertheless, it
%                             still occurs sometimes for accidental
```

```
%                               erroneous input, so it provides a basic
%                               mechanism to check any application using this
%                               algorithm.
%
% This routine can be compiled to increase its speed by a factor of about
% 10-15, which is certainly advisable when the complete application requires
% a great number of Lambert problems to be solved. The entire routine is
% written in embedded MATLAB, so it can be compiled with the emlmex()
% function (older MATLAB) or codegen() function (MATLAB 2011a and later).
%
% To do this using emlmex(), make sure MATLAB's current directory is equal
% to where this file is located. Then, copy-paste and execute the following
% commands to the command window:
%
%    example_input = {...
%         [0.0, 0.0, 0.0], ...% r1vec
%         [0.0, 0.0, 0.0], ...% r2vec
%          0.0, ...           % tf
%          0.0, ...           % m
%          0.0};              % muC
%    emlmex -eg example_input lambert.m
%
% This is of course assuming your compiler is configured correctly. See the
% documentation of emlmex() on how to do that.
%
% Using codegen(), the syntax is as follows:
%
%    example_input = {...
%         [0.0, 0.0, 0.0], ...% r1vec
%         [0.0, 0.0, 0.0], ...% r2vec
%          0.0, ...           % tf
%          0.0, ...           % m
%          0.0};              % muC
%    codegen lambert.m -args example_input
%
% Note that in newer MATLAB versions, the code analyzer will complain about
% the pragma "%#eml" after the main function's name, and possibly, issue
% subsequent warnings related to this issue. To get rid of this problem,
simply
% replace the "%#eml" directive with "%#codegen".
%
%
%
% References:
%
% [1] Izzo, D. ESA Advanced Concepts team. Code used available in MGA.M, on
%     http://www.esa.int/gsp/ACT/inf/op/globopt.htm. Last retreived Nov,
2009.
% [2] Lancaster, E.R. and Blanchard, R.C. "A unified form of Lambert's
theorem."
%     NASA technical note TN D-5368,1969.
% [3] Gooding, R.H. "A procedure for the solution of Lambert's orbital
boundary-value
%     problem. Celestial Mechanics and Dynamical Astronomy, 48:145?165,1990.
```

```
%
% See also lambert_low_ExpoSins.
% Please report bugs and inquiries to:
%
% Name       : Rody P.S. Oldenhuis
% E-mail     : oldenhuis@gmail.com
% Licence    : 2-clause BSD (see License.txt)
% If you find this work useful, please consider a donation:
% https://www.paypal.me/RodyO/3.5
% If you want to cite this work in an academic paper, please use
% the following template:
%
% Rody Oldenhuis, orcid.org/0000-0002-3162-3660. "Lambert" <version>,
% <date you last used it>. MATLAB Robust solver for Lambert's
% orbital-boundary value problem.
% https://nl.mathworks.com/matlabcentral/fileexchange/26348
% ------------------------------------------------------------------
% Izzo's version:
% Very fast, but not very robust for more complicated cases
% ------------------------------------------------------------------
function [V1,V2,extremal_distances,exitflag] =
lambert_izzo_gooding(r1vec,r2vec,tf,m,muC) %#coder
% original documentation:
%{
 This routine implements a new algorithm that solves Lambert's problem. The
 algorithm has two major characteristics that makes it favorable to other
 existing ones.
 1) It describes the generic orbit solution of the boundary condition
 problem through the variable X=log(1+cos(alpha/2)). By doing so the
 graph of the time of flight become defined in the entire real axis and
 resembles a straight line. Convergence is granted within few iterations
 for all the possible geometries (except, of course, when the transfer
 angle is zero). When multiple revolutions are considered the variable is
 X=tan(cos(alpha/2)*pi/2).
 2) Once the orbit has been determined in the plane, this routine
 evaluates the velocity vectors at the two points in a way that is not
 singular for the transfer angle approaching to pi (Lagrange coefficient
 based methods are numerically not well suited for this purpose).
 As a result Lambert's problem is solved (with multiple revolutions
 being accounted for) with the same computational effort for all
 possible geometries. The case of near 180 transfers is also solved
 efficiently.
  We note here that even when the transfer angle is exactly equal to pi
 the algorithm does solve the problem in the plane (it finds X), but it
 is not able to evaluate the plane in which the orbit lies. A solution
 to this would be to provide the direction of the plane containing the
 transfer orbit from outside. This has not been implemented in this
 routine since such a direction would depend on which application the
 transfer is going to be used in.
 please report bugs to dario.izzo@esa.int
%}
% adjusted documentation:
%{
 By default, the short-way solution is computed. The long way solution
```

```
 may be requested by giving a negative value to the corresponding
 time-of-flight [tf].
 For problems with |m| > 0, there are generally two solutions. By
 default, the right branch solution will be returned. The left branch
 may be requested by giving a negative value to the corresponding
 number of complete revolutions [m].
%}
% Authors
% .-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.-`-.
% Name       : Dr. Dario Izzo
% E-mail     : dario.izzo@esa.int
% Affiliation: ESA / Advanced Concepts Team (ACT)
% Made more readible and optimized for speed by Rody P.S. Oldenhuis
% Code available in MGA.M on   http://www.esa.int/gsp/ACT/inf/op/globopt.htm
% last edited 12/Dec/2009
% ADJUSTED FOR EML-COMPILATION 24/Dec/2009
    % initial values
    tol = 1e-14;    bad = false;     days = 86400;
    % work with non-dimensional units

    r1 = sqrt(r1vec*r1vec.');
    r1vec = r1vec/r1;
    V = sqrt(muC/r1);          r2vec = r2vec/r1;
    T = r1/V;                  tf    = tf*days/T; % also transform to seconds
    % relevant geometry parameters (non dimensional)
    mr2vec = sqrt(r2vec*r2vec.');
    % make 100% sure it's in (-1 <= dth <= +1)
    dth = acos( max(-1, min(1, (r1vec*r2vec.')/mr2vec)) );
    % decide whether to use the left or right branch (for multi-revolution
    % problems), and the long- or short way
    leftbranch = sign(m);   longway = sign(tf);
    m = abs(m);             tf = abs(tf);
    if (longway < 0), dth = 2*pi - dth; end
    % derived quantities
    c      = sqrt(1 + mr2vec^2 - 2*mr2vec*cos(dth)); % non-dimensional chord
    s      = (1 + mr2vec + c)/2;                     % non-dimensional semi-
perimeter
    a_min  = s/2;                                    % minimum energy
ellipse semi major axis
    Lambda = sqrt(mr2vec)*cos(dth/2)/s;              % lambda parameter
(from BATTIN's book)
    crossprd = [r1vec(2)*r2vec(3) - r1vec(3)*r2vec(2),...
                r1vec(3)*r2vec(1) - r1vec(1)*r2vec(3),...% non-dimensional
normal vectors
                r1vec(1)*r2vec(2) - r1vec(2)*r2vec(1)];
    mcr       = sqrt(crossprd*crossprd.');           % magnitues thereof
    nrmunit   = crossprd/mcr;                        % unit vector thereof
    % Initial values
    % -------------------------------------------------------------
    % ELMEX requires this variable to be declared OUTSIDE the IF-statement
    logt = log(tf); % avoid re-computing the same value
    % single revolution (1 solution)
    if (m == 0)
        % initial values
```

```
        inn1 = -0.5233;      % first initial guess
        inn2 = +0.5233;      % second initial guess
        x1   = log(1 + inn1);% transformed first initial guess
        x2   = log(1 + inn2);% transformed first second guess
        % multiple revolutions (0, 1 or 2 solutions)
        % the returned soltuion depends on the sign of [m]
    else
        % select initial values
        if (leftbranch < 0)
            inn1 = -0.5234; % first initial guess, left branch
            inn2 = -0.2234; % second initial guess, left branch
        else
            inn1 = +0.7234; % first initial guess, right branch
            inn2 = +0.5234; % second initial guess, right branch
        end
        x1 = tan(inn1*pi/2);% transformed first initial guess
        x2 = tan(inn2*pi/2);% transformed first second guess
    end
    % since (inn1, inn2) < 0, initial estimate is always ellipse
    xx   = [inn1, inn2];  aa = a_min./(1 - xx.^2);
    bbeta = longway * 2*asin(sqrt((s-c)/2./aa));
    % make 100.4% sure it's in (-1 <= xx <= +1)
    aalfa = 2*acos(  max(-1, min(1, xx)) );
    % evaluate the time of flight via Lagrange expression
    y12  = aa.*sqrt(aa).*((aalfa - sin(aalfa)) - (bbeta-sin(bbeta)) +
2*pi*m);
    % initial estimates for y
    if m == 0
        y1 = log(y12(1)) - logt;
        y2 = log(y12(2)) - logt;
    else
        y1 = y12(1) - tf;
        y2 = y12(2) - tf;
    end
    % Solve for x
    % -----------------------------------------------------------
    % Newton-Raphson iterations
    % NOTE - the number of iterations will go to infinity in case
    % m > 0  and there is no solution. Start the other routine in
    % that case
    err = inf;  iterations = 0; xnew = 0;
    while (err > tol)
        % increment number of iterations
        iterations = iterations + 1;
        % new x
        xnew = (x1*y2 - y1*x2) / (y2-y1);
        % copy-pasted code (for performance)
        if m == 0, x = exp(xnew) - 1; else x = atan(xnew)*2/pi; end
        a = a_min/(1 - x^2);
        if (x < 1) % ellipse
            beta = longway * 2*asin(sqrt((s-c)/2/a));
            % make 100.4% sure it's in (-1 <= xx <= +1)
            alfa = 2*acos( max(-1, min(1, x)) );
        else % hyperbola
```

```
            alfa = 2*acosh(x);
            beta = longway * 2*asinh(sqrt((s-c)/(-2*a)));
        end
        % evaluate the time of flight via Lagrange expression
        if (a > 0)
            tof = a*sqrt(a)*((alfa - sin(alfa)) - (beta-sin(beta)) + 2*pi*m);
        else
            tof = -a*sqrt(-a)*((sinh(alfa) - alfa) - (sinh(beta) - beta));
        end
        % new value of y
        if m ==0, ynew = log(tof) - logt; else ynew = tof - tf; end
        % save previous and current values for the next iterarion
        % (prevents getting stuck between two values)
        x1 = x2;  x2 = xnew;
        y1 = y2;  y2 = ynew;
        % update error
        err = abs(x1 - xnew);
        % escape clause
        if (iterations > 15), bad = true; break; end
    end
    % If the Newton-Raphson scheme failed, try to solve the problem
    % with the other Lambert targeter.
    if bad
        % NOTE: use the original, UN-normalized quantities
        [V1, V2, extremal_distances, exitflag] = ...
            lambert_LancasterBlanchard(r1vec*r1, r2vec*r1, longway*tf*T,
leftbranch*m, muC);
        return
    end
    % convert converged value of x
    if m==0, x = exp(xnew) - 1; else x = atan(xnew)*2/pi; end
    %{
      The solution has been evaluated in terms of log(x+1) or tan(x*pi/2), we
      now need the conic. As for transfer angles near to pi the Lagrange-
      coefficients technique goes singular (dg approaches a zero/zero that is
      numerically bad) we here use a different technique for those cases.
When
      the transfer angle is exactly equal to pi, then the ih unit vector is
not
      determined. The remaining equations, though, are still valid.
    %}
    % Solution for the semi-major axis
    a = a_min/(1-x^2);
    % Calculate psi
    if (x < 1) % ellipse
        beta = longway * 2*asin(sqrt((s-c)/2/a));
        % make 100.4% sure it's in (-1 <= xx <= +1)
        alfa = 2*acos( max(-1, min(1, x)) );
        psi  = (alfa-beta)/2;
        eta2 = 2*a*sin(psi)^2/s;
        eta  = sqrt(eta2);
    else       % hyperbola
        beta = longway * 2*asinh(sqrt((c-s)/2/a));
        alfa = 2*acosh(x);
```

```
        psi  = (alfa-beta)/2;
        eta2 = -2*a*sinh(psi)^2/s;
        eta  = sqrt(eta2);
    end
    % unit of the normalized normal vector
    ih = longway * nrmunit;
    % unit vector for normalized [r2vec]
    r2n = r2vec/mr2vec;
    % cross-products
    % don't use cross() (emlmex() would try to compile it, and this way it
    % also does not create any additional overhead)
    crsprd1 = [ih(2)*r1vec(3)-ih(3)*r1vec(2),...
               ih(3)*r1vec(1)-ih(1)*r1vec(3),...
               ih(1)*r1vec(2)-ih(2)*r1vec(1)];
    crsprd2 = [ih(2)*r2n(3)-ih(3)*r2n(2),...
               ih(3)*r2n(1)-ih(1)*r2n(3),...
               ih(1)*r2n(2)-ih(2)*r2n(1)];
    % radial and tangential directions for departure velocity
    Vr1 = 1/eta/sqrt(a_min) * (2*Lambda*a_min - Lambda - x*eta);
    Vt1 = sqrt(mr2vec/a_min/eta2 * sin(dth/2)^2);
    % radial and tangential directions for arrival velocity
    Vt2 = Vt1/mr2vec;
    Vr2 = (Vt1 - Vt2)/tan(dth/2) - Vr1;
    % terminal velocities
    V1 = (Vr1*r1vec + Vt1*crsprd1)*V;
    V2 = (Vr2*r2n + Vt2*crsprd2)*V;
    % exitflag
    exitflag = 1; % (success)
    % also compute minimum distance to central body
    % NOTE: use un-transformed vectors again!
    extremal_distances = ...
        minmax_distances(r1vec*r1, r1, r2vec*r1, mr2vec*r1, dth, a*r1, V1,
V2, m, muC);
end
% -----------------------------------------------------------------
% Lancaster & Blanchard version, with improvements by Gooding
% Very reliable, moderately fast for both simple and complicated cases
% -----------------------------------------------------------------
function [V1,...
          V2,...
          extremal_distances,...
          exitflag] = lambert_LancasterBlanchard(r1vec,...
                                                 r2vec,...
                                                 tf,...
                                                 m,...
                                                 muC) %#coder
%{
LAMBERT_LANCASTERBLANCHARD       High-Thrust Lambert-targeter
lambert_LancasterBlanchard() uses the method developed by
Lancaster & Blancard, as described in their 1969 paper. Initial values,
and several details of the procedure, are provided by R.H. Gooding,
as described in his 1990 paper.
%}
% Please report bugs and inquiries to:
```

```
%
% Name        : Rody P.S. Oldenhuis
% E-mail      : oldenhuis@gmail.com
% Licence     : 2-clause BSD (see License.txt)
% If you find this work useful, please consider a donation:
% https://www.paypal.me/RodyO/3.5
    % ADJUSTED FOR EML-COMPILATION 29/Sep/2009
    % manipulate input
    tol     = 1e-12;                            % optimum for numerical
noise v.s. actual precision
    r1      = sqrt(r1vec*r1vec.');              % magnitude of r1vec
    r2      = sqrt(r2vec*r2vec.');              % magnitude of r2vec
    r1unit  = r1vec/r1;                         % unit vector of r1vec
    r2unit  = r2vec/r2;                         % unit vector of r2vec
    crsprod = cross(r1vec, r2vec, 2);           % cross product of r1vec and
r2vec
    mcrsprd = sqrt(crsprod*crsprod.');          % magnitude of that cross
product
    th1unit = cross(crsprod/mcrsprd, r1unit);   % unit vectors in the
tangential-directions
    th2unit = cross(crsprod/mcrsprd, r2unit);
    % make 100.4% sure it's in (-1 <= x <= +1)
    dth = acos( max(-1, min(1, (r1vec*r2vec.')/r1/r2)) ); % turn angle
    % if the long way was selected, the turn-angle must be negative
    % to take care of the direction of final velocity
    longway = sign(tf); tf = abs(tf);
    if (longway < 0), dth = dth-2*pi; end
    % left-branch
    leftbranch = sign(m); m = abs(m);
    % define constants
    c  = sqrt(r1^2 + r2^2 - 2*r1*r2*cos(dth));
    s  = (r1 + r2 + c) / 2;
    T  = sqrt(8*muC/s^3) * tf;
    q  = sqrt(r1*r2)/s * cos(dth/2);
    % general formulae for the initial values (Gooding)
    % ------------------------------------------------
    % some initial values
    T0  = LancasterBlanchard(0, q, m);
    Td  = T0 - T;
    phr = mod(2*atan2(1 - q^2, 2*q), 2*pi);
    % initial output is pessimistic
    V1 = NaN(1,3);    V2 = V1;    extremal_distances = [NaN, NaN];
    % single-revolution case
    if (m == 0)
        x01 = T0*Td/4/T;
        if (Td > 0)
            x0 = x01;
        else
            x01 = Td/(4 - Td);
            x02 = -sqrt( -Td/(T+T0/2) );
            W   = x01 + 1.7*sqrt(2 - phr/pi);
            if (W >= 0)
                x03 = x01;
            else
```

```
                x03 = x01 + (-W).^(1/16).*(x02 - x01);
            end
            lambda = 1 + x03*(1 + x01)/2 - 0.03*x03^2*sqrt(1 + x01);
            x0 = lambda*x03;
        end
        % this estimate might not give a solution
        if (x0 < -1), exitflag = -1; return; end
    % multi-revolution case
    else
        % determine minimum Tp(x)
        xMpi = 4/(3*pi*(2*m + 1));
        if (phr < pi)
            xM0 = xMpi*(phr/pi)^(1/8);
        elseif (phr > pi)
            xM0 = xMpi*(2 - (2 - phr/pi)^(1/8));
        % EMLMEX requires this one
        else
            xM0 = 0;
        end
        % use Halley's method
        xM = xM0;  Tp = inf;  iterations = 0;
        while abs(Tp) > tol
            % iterations
            iterations = iterations + 1;
            % compute first three derivatives
            [dummy, Tp, Tpp, Tppp] = LancasterBlanchard(xM, q, m);%#ok
            % new value of xM
            xMp = xM;
            xM  = xM - 2*Tp.*Tpp ./ (2*Tpp.^2 - Tp.*Tppp);
            % escape clause
            if mod(iterations, 7), xM = (xMp+xM)/2; end
            % the method might fail. Exit in that case
            if (iterations > 25), exitflag = -2; return; end
        end
        % xM should be elliptic (-1 < x < 1)
        % (this should be impossible to go wrong)
        if (xM < -1) || (xM > 1), exitflag = -1; return; end
        % corresponding time
        TM = LancasterBlanchard(xM, q, m);
        % T should lie above the minimum T
        if (TM > T), exitflag = -1; return; end
        % find two initial values for second solution (again with lambda-
type patch)
        %
-------------------------------------------------------------------------
        % some initial values
        TmTM  = T - TM;   T0mTM = T0 - TM;
        [dummy, Tp, Tpp] = LancasterBlanchard(xM, q, m);%#ok
        % first estimate (only if m > 0)
        if leftbranch > 0
            x   = sqrt( TmTM / (Tpp/2 + TmTM/(1-xM)^2) );
            W   = xM + x;
            W   = 4*W/(4 + TmTM) + (1 - W)^2;
            x0  = x*(1 - (1 + m + (dth - 1/2)) / ...
```

```
                (1 + 0.15*m)*x*(W/2 + 0.03*x*sqrt(W))) + xM;
            % first estimate might not be able to yield possible solution
            if (x0 > 1), exitflag = -1; return; end
        % second estimate (only if m > 0)
        else
            if (Td > 0)
                x0 = xM - sqrt(TM/(Tpp/2 - TmTM*(Tpp/2/T0mTM - 1/xM^2)));
            else
                x00 = Td / (4 - Td);
                W = x00 + 1.7*sqrt(2*(1 - phr));
                if (W >= 0)
                    x03 = x00;
                else
                    x03 = x00 - sqrt((-W)^(1/8))*(x00 + sqrt(-Td/(1.5*T0 -
Td)));
                end
                W      = 4/(4 - Td);
                lambda = (1 + (1 + m + 0.24*(dth - 1/2)) / ...
                    (1 + 0.15*m)*x03*(W/2 - 0.03*x03*sqrt(W)));
                x0     = x03*lambda;
            end
            % estimate might not give solutions
            if (x0 < -1), exitflag = -1; return; end
        end
    end
    % find root of Lancaster & Blancard's function
    % --------------------------------------------
    % (Halley's method)
    x = x0; Tx = inf; iterations = 0;
    while abs(Tx) > tol
        % iterations
        iterations = iterations + 1;
        % compute function value, and first two derivatives
        [Tx, Tp, Tpp] = LancasterBlanchard(x, q, m);
        % find the root of the *difference* between the
        % function value [T_x] and the required time [T]
        Tx = Tx - T;
        % new value of x
        xp = x;
        x  = x - 2*Tx*Tp ./ (2*Tp^2 - Tx*Tpp);
        % escape clause
        if mod(iterations, 7), x = (xp+x)/2; end
        % Halley's method might fail
        if iterations > 25, exitflag = -2; return; end
    end
    % calculate terminal velocities
    % -----------------------------
    % constants required for this calculation
    gamma = sqrt(muC*s/2);
    if (c == 0)
        sigma = 1;
        rho   = 0;
        z     = abs(x);
    else
```

```
        sigma = 2*sqrt(r1*r2/(c^2)) * sin(dth/2);
        rho   = (r1 - r2)/c;
        z     = sqrt(1 + q^2*(x^2 - 1));
    end
    % radial component
    Vr1    = +gamma*((q*z - x) - rho*(q*z + x)) / r1;
    Vr1vec = Vr1*r1unit;
    Vr2    = -gamma*((q*z - x) + rho*(q*z + x)) / r2;
    Vr2vec = Vr2*r2unit;
    % tangential component
    Vtan1    = sigma * gamma * (z + q*x) / r1;
    Vtan1vec = Vtan1 * th1unit;
    Vtan2    = sigma * gamma * (z + q*x) / r2;
    Vtan2vec = Vtan2 * th2unit;
    % Cartesian velocity
    V1 = Vtan1vec + Vr1vec;
    V2 = Vtan2vec + Vr2vec;
    % exitflag
    exitflag = 1; % (success)
    % also determine minimum/maximum distance
    a = s/2/(1 - x^2); % semi-major axis
    extremal_distances = minmax_distances(r1vec, r1, r1vec, r2, dth, a, V1,
V2, m, muC);
end
% Lancaster & Blanchard's function, and three derivatives thereof
function [T, Tp, Tpp, Tppp] = LancasterBlanchard(x, q, m)
    % protection against idiotic input
    if (x < -1) % impossible; negative eccentricity
        x = abs(x) - 2;
    elseif (x == -1) % impossible; offset x slightly
        x = x + eps;
    end
    % compute parameter E
    E  = x*x - 1;
    % T(x), T'(x), T''(x)
    if x == 1 % exactly parabolic; solutions known exactly
        % T(x)
        T = 4/3*(1-q^3);
        % T'(x)
        Tp = 4/5*(q^5 - 1);
        % T''(x)
        Tpp = Tp + 120/70*(1 - q^7);
        % T'''(x)
        Tppp = 3*(Tpp - Tp) + 2400/1080*(q^9 - 1);
    elseif abs(x-1) < 1e-2 % near-parabolic; compute with series
        % evaluate sigma
        [sig1, dsigdx1, d2sigdx21, d3sigdx31] = sigmax(-E);
        [sig2, dsigdx2, d2sigdx22, d3sigdx32] = sigmax(-E*q*q);
        % T(x)
        T = sig1 - q^3*sig2;
        % T'(x)
        Tp = 2*x*(q^5*dsigdx2 - dsigdx1);
        % T''(x)
        Tpp = Tp/x + 4*x^2*(d2sigdx21 - q^7*d2sigdx22);
```

```
        % T'''(x)
        Tppp = 3*(Tpp-Tp/x)/x + 8*x*x*(q^9*d3sigdx32 - d3sigdx31);
    else % all other cases
        % compute all substitution functions
        y  = sqrt(abs(E));
        z  = sqrt(1 + q^2*E);
        f  = y*(z - q*x);
        g  = x*z - q*E;
        % BUGFIX: (Simon Tardivel) this line is incorrect for E==0 and f+g==0
        % d  = (E < 0)*(atan2(f, g) + pi*m) + (E > 0)*log( max(0, f + g) );
        % it should be written out like so:
        if (E<0)
            d = atan2(f, g) + pi*m;
        elseif (E==0)
            d = 0;
        else
            d = log(max(0, f+g));
        end
        % T(x)
        T = 2*(x - q*z - d/y)/E;
        %  T'(x)
        Tp = (4 - 4*q^3*x/z - 3*x*T)/E;
        % T''(x)
        Tpp = (-4*q^3/z * (1 - q^2*x^2/z^2) - 3*T - 3*x*Tp)/E;
        % T'''(x)
        Tppp = (4*q^3/z^2*((1 - q^2*x^2/z^2) + 2*q^2*x/z^2*(z - x)) - 8*Tp -
7*x*Tpp)/E;
    end
end
% series approximation to T(x) and its derivatives
% (used for near-parabolic cases)
function [sig, dsigdx, d2sigdx2, d3sigdx3] = sigmax(y)
    % preload the factors [an]
    % (25 factors is more than enough for 16-digit accuracy)
    persistent an;
    if isempty(an)
        an = [
            4.000000000000000e-001;     2.142857142857143e-001;
4.629629629629630e-002
            6.628787878787879e-003;     7.211538461538461e-004;
6.365740740740740e-005
            4.741479925303455e-006;     3.059406328320802e-007;
1.742836409255060e-008
            8.892477331109578e-010;     4.110111531986532e-011;
1.736709384841458e-012
            6.759767240041426e-014;     2.439123386614026e-015;
8.203411614538007e-017
            2.583771576869575e-018;     7.652331327976716e-020;
2.138860629743989e-021
            5.659959451165552e-023;     1.422104833817366e-024;
3.401398483272306e-026
            7.762544304774155e-028;     1.693916882090479e-029;
3.541295006766860e-031
            7.105336187804402e-033];
```

```
    end
    % powers of y
    powers = y.^(1:25);
    % sigma itself
    sig = 4/3 + powers*an;
    % dsigma / dx (derivative)
    dsigdx = ( (1:25).*[1, powers(1:24)] ) * an;
    % d2sigma / dx2 (second derivative)
    d2sigdx2 = ( (1:25).*(0:24).*[1/y, 1, powers(1:23)] ) * an;
    % d3sigma / dx3 (third derivative)
    d3sigdx3 = ( (1:25).*(0:24).*(-1:23).*[1/y/y, 1/y, 1, powers(1:22)] ) *
an;
end
% ------------------------------------------------------------------
% Helper functions
% ------------------------------------------------------------------
% compute minimum and maximum distances to the central body
function extremal_distances = minmax_distances(r1vec, r1,...
                                               r2vec, r2,...
                                               dth,...
                                               a,...
                                               V1, V2,...
                                               m,...
                                               muC)
    % default - minimum/maximum of r1,r2
    minimum_distance = min(r1,r2);
    maximum_distance = max(r1,r2);
    % was the longway used or not?
    longway = abs(dth) > pi;
    % eccentricity vector (use triple product identity)
    evec = ((V1*V1.')*r1vec - (V1*r1vec.')*V1)/muC - r1vec/r1;
    % eccentricity
    e = sqrt(evec*evec.');
    % apses
    pericenter = a*(1-e);
    apocenter  = inf;                    % parabolic/hyperbolic case
    if (e < 1), apocenter = a*(1+e); end % elliptic case
    % since we have the eccentricity vector, we know exactly where the
    % pericenter lies. Use this fact, and the given value of [dth], to
    % cross-check if the trajectory goes past it
    if (m > 0) % obvious case (always elliptical and both apses are traversed)
        minimum_distance = pericenter;
        maximum_distance = apocenter;
    else % less obvious case
        % compute theta1&2 ( use (AxB)-(CxD) = (C·B)(D·A) - (C·A)(B·D) ))
        pm1 = sign( r1*r1*(evec*V1.') - (r1vec*evec.')*(r1vec*V1.') );
        pm2 = sign( r2*r2*(evec*V2.') - (r2vec*evec.')*(r2vec*V2.') );
        % make 100.4% sure it's in (-1 <= theta12 <= +1)
        theta1 = pm1*acos( max(-1, min(1, (r1vec/r1)*(evec/e).')) );
        theta2 = pm2*acos( max(-1, min(1, (r2vec/r2)*(evec/e).')) );
        % points 1&2 are on opposite sides of the symmetry axis -- minimum
        % and maximum distance depends both on the value of [dth], and both
        % [theta1] and [theta2]
        if (theta1*theta2 < 0)
```

```
            % if |th1| + |th2| = turnangle, we know that the pericenter was
            % passed
            if abs(abs(theta1) + abs(theta2) - dth) < 5*eps(dth)
                minimum_distance = pericenter;
            % this condition can only be false for elliptic cases, and
            % when it is indeed false, we know that the orbit passed
            % apocenter
            else
                maximum_distance = apocenter;
            end
        % points 1&2 are on the same side of the symmetry axis. Only if the
        % long-way was used are the min. and max. distances different from
        % the min. and max. values of the radii (namely, equal to the apses)
        elseif longway
            minimum_distance = pericenter;
            if (e < 1), maximum_distance = apocenter; end
        end
    end
    % output argument
    extremal_distances = [minimum_distance, maximum_distance];
end
```

Published with MATLAB® R2023b

```
function [V0,V1] = lamberts_Gauss(R0,R1,deltat)

% Description.
% Find velocity at two points of known position, ECI. Geometric.
% This follows Vallado and lecture notes.
% trig and algebra to get to this point; see lecture notes

mu = 398600; % km3/s2

% .... Find delta nu
tm = 1; % short way
r0 = norm(R0);
r1 = norm(R1);
cosDeltaNu = dot(R0,R1)/(r0*r1);
sinDeltaNu = tm*sqrt(1 - cosDeltaNu^2); % in rad
dNu = asin(sinDeltaNu); % in rad

y = 1; % initial guess
err = 1;
tol = 1e-6;
nmax = 1000;
n = 0;
l = ( (r0 + r1) / ( 4*sqrt(r0*r1)*cos(dNu/2))  ) - 1/2;
m = mu*deltat^2 / ( 2*sqrt(r0*r1)* cos(dNu/2) )^3;

while err > tol && n < nmax
    y0 = y;
    x = m/y0^2 - l;
    X = 4/3 * (1 + (6*x)/5 + (48*x^2)/35 + (480*x^3)/315); % series expansion
    y = 1 + X*(l+x);
    err = abs(y - y0);
    n = n + 1;
end

% Use Gauss equation (6) to find deltaE
dE = 4*asin(sqrt(x)); % in rad

% Use Gauss equation (7) to find semi-major axis, a
a.gauss = ( (sqrt(mu) * deltat) / (2 * y * sqrt(r0*r1) * sin(dE/2)  *
cos(dNu/2)) )^ (2); % in km

% dr A lecture
% Use f, g to find velocity.
    f = 1 - a.gauss/r1 * (1 - cos(dE));
    g = deltat - (a.gauss^(3/2)) / sqrt(mu) * (dE - sin(dE));

% Vallado (for elliptical orbits)
    num = r0*r1*(1-cosDeltaNu);
    denom = r0 + r1 - 2*sqrt(r0*r1)*cos(dNu/2)*cos(dE/2);
    p = num/denom;
    % f = 1 - r1/p*(1-cosDeltaNu);
    % g = r1*r0*sin(dNu)/ sqrt(mu*p);
```

```
    gdot = 1 - r0/p * (1 - cosDeltaNu);

% Solve for velocity at first observation
V0 = (R1 - f*R0) / g;
V1 = (gdot*R1 - R0)/g;
```

Published with MATLAB® R2023b

```
function [V1,V2] = lamberts_universalVar(R1,R2,t,tm)

% This function takes two position vectors and the time between them as
% inputs and outputs velocity vectors v1 and v2 corresponding to the
% necessary velocities to traverse the desired path.
```

Lambert's Problem

```
mu = 398600; % km3/s2
r1vect = R1;
r2vect = R2;

r1 = norm(r1vect);
r2 = norm(r2vect);

% Find delta theta
cross12 = cross(r1vect,r2vect);
zcross = cross12(3);
dtheta = acos(dot(r1vect,r2vect) / (r1*r2));

% Check for pro/retro

if tm == 1                                % PROGRADE ASSUMPTION

        if zcross < 0
            dtheta = 2*pi - dtheta;
            disp("Prograde orbit assumed.")
        end

    elseif tm == -1                         % RETROGRADE ASSUMPTION
        if zcross >= 0
            dtheta = 2*pi - dtheta;
            disp("Retrograde orbit assumed.")
        end
else
    error("tm must be + or - 1")

end % big if statement

% Calculate A using Equation 5.35
A = sin(dtheta) * sqrt( (r1*r2)/(1 - cos(dtheta)) );

% Use Eqns. (5.40), (5.43), and (5.45) to solve (5.39) for z.
% Use Newtons interative solver.

% Rest is from CURTIS:
%...Determine approximately where F(z,t) changes sign, and
%...use that value of z as the starting value for Equation 5.45:
z = -100;
while F(z,t) < 0
z = z + 0.1;
end
```

```
%...Set an error tolerance and a limit on the number of iterations:
tol = 1.e-8;
nmax = 5000;

%...Iterate on Equation 5.45 until z is determined to within the
%...error tolerance:
ratio = 1;
n = 0;
while (abs(ratio) > tol) & (n <= nmax)
    n = n+1;
    ratio = F(z,t) / dFdz(z);
    z = z - ratio;
end

if n == nmax
    disp("Maximum number of iterations reached.")
end

%...Equation 5.46a:
f = 1 - y(z)/r1;

%...Equation 5.46b:
g = A*sqrt(y(z)/mu);

%...Equation 5.46d:
gdot = 1 - y(z)/r2;

%...Equation 5.28:
V1 = 1/g*(R2 - f*R1);

%...Equation 5.29:
V2 = 1/g*(gdot*R2 - R1);

%
% Subfunctions used in the main body:
%
%...Equation 5.38:
function dum = y(z)
dum = r1 + r2 + A*(z*S(z) - 1)/sqrt(C(z));
end

%...Equation 5.40:
function dum = F(z,t)
dum = (y(z)/C(z))^1.5*S(z) + A*sqrt(y(z)) - sqrt(mu)*t;
end

%...Equation 5.43:
function dum = dFdz(z)
if z == 0
dum = sqrt(2)/40*y(0)^1.5 + A/8*(sqrt(y(0)) + A*sqrt(1/2/y(0)));
else
dum = (y(z)/C(z))^1.5*(1/2/z*(C(z) - 3*S(z)/2/C(z)) ...
+ 3*S(z)^2/4/C(z)) + A/8*(3*S(z)/C(z)*sqrt(y(z)) ...
+ A*sqrt(C(z)/y(z)));
```

```
end
end

%...Stumpff functions:
function dum = C(z)
dum = Cz(z);
end

function dum = S(z)
dum = Sz(z);
end

function C = Cz(z)

% This function takes alpha and X as part of the universal variable solving
% algorithm.

C = (1/2) - (z/24) + ((z^2) / 720) - ((z^3) / 40320); % enough!

end % C

function [output] = Sz(z)
% S(z) Stumpff Function
% Defined by infinte series

output = (1/6)-(z/120)+((z^2)/5040)-((z^3)/362880);

end % S

end % lamberts solver UV
```

Published with MATLAB® R2023b

```
function [V0,V1,deltatm] = lamberts_minimumEnergy(R0,R1,deltat,tm)

    mu = 398600; % km3/s2

    % chord length; law of cosines
    r0 = norm(R0);
    r1 = norm(R1);

    cosDeltaNu = dot(R0,R1)/(r0*r1);
    sinDeltaNu = tm*sqrt(1 - cosDeltaNu^2); % in rad
    dNu = asin(sinDeltaNu); % in rad

    c = sqrt(r0^2 + r1^2 - 2*r0*r1*cosDeltaNu); % km

    % semiperimeter, s = half the sum of the sides of the triangle (see
Vallado
    % fig. 7-10)
    s = (r0 + r1 + c) / 2;

    % minimum energy semi major axis
    am = s/2;

    % Min ellipse energy
    emin = (r1 - r0) / c;
    pmin = r0*r1/c * (1-cosDeltaNu); % vallado algorithm uses this

    % minimum energy cases
    beta = 2*asin(sqrt( (s-c)/s));
    alpha = 2*asin(sqrt(s/2/am));

    % Check quadrant
    if dNu >= pi
        beta = -beta;
    end

    % find minimum time
    if tm == 1 % short way
        deltatm = am^(3/2) / (sqrt(mu)) * ((alpha - sin(alpha)) - (beta -
sin(beta)));
        elseif tm == -1 % long way
        deltatm = am^(3/2) / (sqrt(mu)) * ((alpha - sin(alpha)) + (beta -
sin(beta)));
        else
        warning("tm must be -1 or 1")
    end

    if deltat <= deltatm
        alpha = pi;
        else
        alpha = 2*pi - pi;
    end
```

```
    % Check bounds; find parabolic solution
    deltat_parabolic = sqrt(2)/3 /sqrt(mu) * ( s^(3/2) - (s-c)^(3/2));

    if deltat_parabolic > deltat
        warning("Delta time given satisfies PARABOLIC SOLUTION.")
    end

    % Use deltaTM as an input to Lamberts to find both velocities
    [V0,V1] = lamberts_universalVar(R0,R1,deltatm,tm);

end % lamberts minimum energy
```

Published with MATLAB® R2023b