```
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function [coe, r, v, jd] = curtisPlanet_elements_and_sv ...
    (planet_id, year, month, day, hour, minute, second)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%{
This function calculates the orbital elements and the state
vector of a planet from the date (year, month, day)
and universal time (hour, minute, second).
mu - gravitational parameter of the sun (km^3/s^2)
deg - conversion factor between degrees and radians
pi - 3.1415926...
coe - vector of heliocentric orbital elements
[h e RA incl w TA a w_hat L M E],
where
h = angular momentum (km^2/s)
e = eccentricity
RA = right ascension (deg)
incl = inclination (deg)
w = argument of perihelion (deg)
TA = true anomaly (deg)
a = semimajor axis (km)
w_hat = longitude of perihelion ( = RA + w) (deg)
L = mean longitude ( = w_hat + M) (deg)
M = mean anomaly (deg)
E = eccentric anomaly (deg)
planet_id - planet identifier:
1 = Mercury
2 = Venus
3 = Earth
4 = Mars
5 = Jupiter
7 = Uranus
8 = Neptune
9 = Pluto
year - range: 1901 - 2099
month - range: 1 - 12
day - range: 1 - 31
hour - range: 0 - 23
minute - range: 0 - 60
second - range: 0 - 60
j0 - Julian day number of the date at 0 hr UT
ut - universal time in fractions of a day
jd - julian day number of the date and time
J2000_coe - row vector of J2000 orbital elements from Table 9.1
rates - row vector of Julian centennial rates from Table 9.1
t0 - Julian centuries between J2000 and jd
elements - orbital elements at jd
r - heliocentric position vector
v - heliocentric velocity vector
User M-functions required: J0, kepler_E, sv_from_coe
User subfunctions required: planetary_elements, zero_to_360
%}
```

```
% ----------------------------------------------------------------------
mu = 132712440018;
deg = pi/180;
%...Equation 5.48:

j0 = 367*year - fix(7*(year + fix((month + 9)/12))/4)+ fix(275*month/9) + day
 + 1721013.5;

ut = (hour + minute/60 + second/3600)/24;
%...Equation 5.47
jd = j0 + ut;
%...Obtain the data for the selected planet from Table 8.1:
[J2000_coe, rates] = planetary_elements(planet_id);
%...Equation 8.93a:
t0 = (jd - 2451545)/36525;
%...Equation 8.93b:
elements = J2000_coe + rates*t0;
a = elements(1);
e = elements(2);
%...Equation 2.71:
h = sqrt(mu*a*(1 - e^2));
%...Reduce the angular elements to within the range 0 - 360 degrees:
incl = elements(3);
RA = zero_to_360(elements(4));
w_hat = zero_to_360(elements(5));
L = zero_to_360(elements(6));
w = zero_to_360(w_hat - RA);
M = zero_to_360((L - w_hat));
%...Algorithm 3.1 (for which M must be in radians)
% E = kepler_E(e, M*deg);
[~,~,E] = joshAnomalyCalculator(e,M,'Me');

%...Equation 3.13 (converting the result to degrees):
TA = zero_to_360...
    (2*atan(sqrt((1 + e)/(1 - e))*tan(E/2))/deg);
coe = [h e RA incl w TA a w_hat L M E/deg];
%...Algorithm 4.5 (for which all angles must be in radians):

% testing my COE -> state vectors
% [r, v] = curtisSv_from_coe([h e RA*deg incl*deg w*deg TA*deg],mu)
[r,v] = joshCOE2rv(a,e,deg2rad(TA),deg2rad(incl),deg2rad(RA),deg2rad(w),mu);


return
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    function [J2000_coe, rates] = planetary_elements(planet_id)
        % ------------------------------------------------------
        %{
This function extracts a planet's J2000 orbital elements and
centennial rates from Table 8.1.
planet_id - 1 through 9, for Mercury through Pluto
J2000_elements - 9 by 6 matrix of J2000 orbital elements for the nine
planets Mercury through Pluto. The columns of each
row are:
```

```
a = semimajor axis (AU)
e = eccentricity
i = inclination (degrees)
RA = right ascension of the ascending
node (degrees)
w_hat = longitude of perihelion (degrees)
L = mean longitude (degrees)
cent_rates - 9 by 6 matrix of the rates of change of the
J2000_elements per Julian century (Cy). Using "dot"
for time derivative, the columns of each row are:
a_dot (AU/Cy)
e_dot (1/Cy)
i_dot (arcseconds/Cy)
RA_dot (arcseconds/Cy)
w_hat_dot (arcseconds/Cy)
Ldot (arcseconds/Cy)
J2000_coe - row vector of J2000_elements corresponding
to "planet_id", with au converted to km
rates - row vector of cent_rates corresponding to
"planet_id", with au converted to km and
arcseconds converted to degrees
au - astronomical unit (km)
        %}
        % ------------------------------------------------------------------------
        J2000_elements = ...
            [ 0.38709893 0.20563069 7.00487 48.33167 77.45645 252.25084
            0.72333199 0.00677323 3.39471 76.68069 131.53298 181.97973
            1.00000011 0.01671022 0.00005 -11.26064 102.94719 100.46435
            1.52366231 0.09341233 1.85061 49.57854 336.04084 355.45332
            5.20336301 0.04839266 1.30530 100.55615 14.75385 34.40438
            9.53707032 0.05415060 2.48446 113.71504 92.43194 49.94432
            19.19126393 0.04716771 0.76986 74.22988 170.96424 313.23218
            30.06896348 0.00858587 1.76917 131.72169 44.97135 304.88003
            39.48168677 0.24880766 17.14175 110.30347 224.06676 238.92881];
        cent_rates = ...
            [ 0.00000066 0.00002527 -23.51 -446.30 573.57 538101628.29
            0.00000092 -0.00004938 -2.86 -996.89 -108.80 210664136.06
            -0.00000005 -0.00003804 -46.94 -18228.25 1198.28 129597740.63
            -0.00007221 0.00011902 -25.47 -1020.19 1560.78 68905103.78
            0.00060737 -0.00012880 -4.15 1217.17 839.93 10925078.35
            -0.00301530 -0.00036762 6.11 -1591.05 -1948.89 4401052.95
            0.00152025 -0.00019150 -2.09 -1681.4 1312.56 1542547.79
            -0.00125196 0.00002514 -3.64 -151.25 -844.43 786449.21
            -0.00076912 0.00006465 11.07 -37.33 -132.25 522747.90];
        J2000_coe = J2000_elements(planet_id,:);
        rates = cent_rates(planet_id,:);
        %...Convert from AU to km:
        au = 149597871;
        J2000_coe(1) = J2000_coe(1)*au;
        rates(1) = rates(1)*au;
        %...Convert from arcseconds to fractions of a degree:
        rates(3:6) = rates(3:6)/3600;
    end %planetary_elements
% ~~~~~~~~~~~~~~~~~~~~~~
```

```
    function y = zero_to_360(x)
        % ~~~~~~~~~~~~~~~~~~~~~~
        %{
This function reduces an angle to lie in the range 0 - 360 degrees.
x - the original angle in degrees
y - the angle reduced to the range 0 - 360 degrees
        %}
        % --------------------------
        if x >= 360
            x = x - fix(x/360)*360;
        elseif x < 0
            x = x - (fix(x/360) - 1)*360;
        end
        y = x;
    end %zero_to_360
end %planet_elements_and_sv
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Published with MATLAB® R2022a