ssb

May 29, 2021

William Daniel Taylor

Space Science with Python: Part 2

Kepler's First Law

Kepler's first law of planetary motion is this:
The orbit of a planet is an ellipse with the Sun at one of two foci.
This exercise explores the Solar System Barycenter (SSB).

[1]:
```
# last tutorial loaded two or three kernels, and projects often require more
# SPICE allows the use of a "kernel meta file" that we can furnish to load
# every kernel we need
import datetime
import spiceypy
import numpy as np
from matplotlib import pyplot as plt

spiceypy.furnsh("kernel_meta.txt")
```

This is a position computation, so we need a start date-time and end date-time.
Like last tutorial, use Python's `datetime` then convert to Ephemeris time.

[2]:
```
# we want the SSB w.r.t. the center of the sun for a certain time interval

# set initial UTC time to 1.1.2000 at midnight
INIT_TIME_UTC = datetime.datetime(year = 2000, month = 1, day = 1,
                                  hour = 0, minute = 0, second = 0)

# add a number of days to this
delta_days = 10000
END_TIME_UTC = INIT_TIME_UTC + datetime.timedelta(days = delta_days)

# convert date-times to strings
INIT_TIME_UTC_STR = INIT_TIME_UTC.strftime("%Y-%m-%d %H:%M:%S")
END_TIME_UTC_STR = END_TIME_UTC.strftime("%Y-%m-%d %H:%M:%S")

# print start and end times
print("init time (UTC): %s" % INIT_TIME_UTC_STR)
```

```python
print("end time (UTC): %s" % END_TIME_UTC_STR)

# convert to ET using SPICE
INIT_TIME_ET = spiceypy.utc2et(INIT_TIME_UTC_STR)
END_TIME_ET = spiceypy.utc2et(END_TIME_UTC_STR)
```

```
init time (UTC): 2000-01-01 00:00:00
end time (UTC): 2027-05-19 00:00:00
```

Note of precision: in some scientific measurements accounting for leap-seconds is crucial. The `utc2etc()` function takes care of this.

[3]:
```python
# a day has 86,000 seconds (24 hrs * 60 min * 60 sec)
# this tutorial has a time period of 10,000 days
# thus, the difference in seconds between the start time and end time should be
# 10,000 * 86,400
# let's look at the delta

print("time covered in seconds: %s" % (END_TIME_ET - INIT_TIME_ET))

# the utc2et function added 5.0012845 seconds (leapseconds)!
```

```
time covered in seconds: 864000005.0012845
```

Now we need an array that contains `10,000` time steps between the start and end times.

[4]:
```python
time_intervals_et = np.linspace(INIT_TIME_ET, END_TIME_ET, delta_days)
```

Now we'll compute the SSB position in the x, y, z direction.
Use `spkgps` with paremeters: `targ = NAIFID, et, ref = reference, obs = 10`.
returns the position of the SSB w.r.t the Sun at the given ET, as well as the light time.
We don't need light time, so we use a single underscore.

[5]:
```python
# we need an empty list to store all x, y, z components for each of our
# 10,000 time steps
ssb_wrt_sun_position = []

# populate this list for each ET time interval we generated above
# we use ECLIPJ2000 like in the previous tutorial
for time_intervals_et_f in time_intervals_et:
    _position, _ = spiceypy.spkgps(targ = 0, et = time_intervals_et_f,
                                   ref = "ECLIPJ2000", obs = 10)

    # append the result to the list
    ssb_wrt_sun_position.append(_position)

# convert the list to a numpy array
ssb_wrt_sun_position = np.array(ssb_wrt_sun_position)
```

[6]:
```python
# take a look at the initial time
print("Position components of the Solar System Barycenter w.r.t. the\n" +
      "center of the Sun (at t = 0): \n" +
      "X = %s km\n Y = %s km\n Z = %s km\n" %
      tuple(np.round(ssb_wrt_sun_position[0])))

# let's compute the corresponding distance using numpy's linalg.norm()
print("Distance to the SSB w.r.t. the\n" +
      "center of the Sun (at t = 0): \n d = %s km"
      % round(np.linalg.norm(ssb_wrt_sun_position[0])))
```

```
Position components of the Solar System Barycenter w.r.t. the
center of the Sun (at t = 0):
X = 1068000.0 km
 Y = 417681.0 km
 Z = -30845.0 km

Distance to the SSB w.r.t. the
center of the Sun (at t = 0):
 d = 1147185 km
```

These are astronomically large numbers (see what I did there?).
To sanity-check, let's define a scale by using the radius of the Sun,
given by the corresponding SPICE kernel.

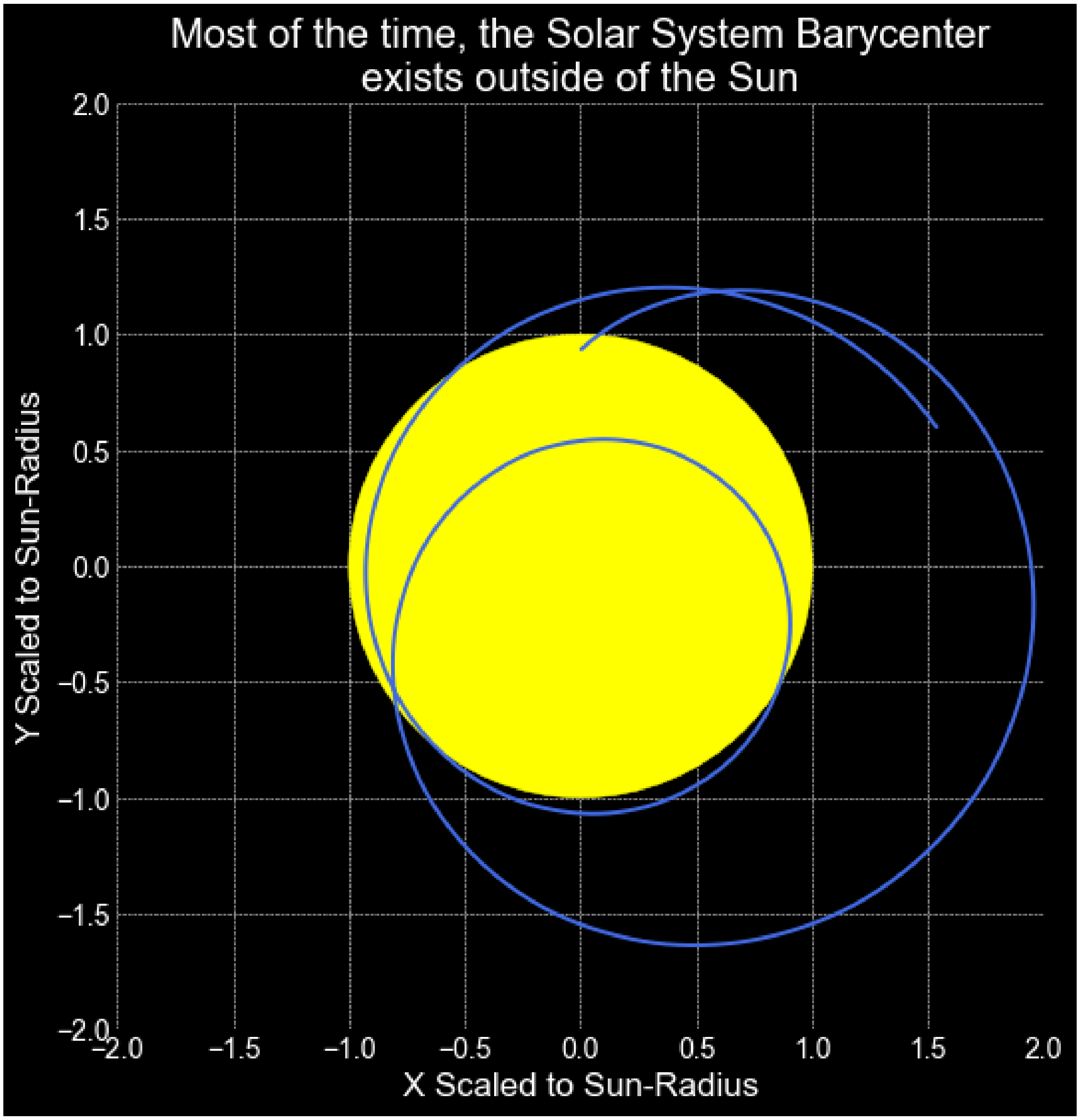
[7]:
```python
# our next goal is to plot the movement. is it interesting?

# km is too small, AU is too large. let's scale according to the radius of the␣
↪sun
# extract the Sun radii (x, y, z components of the Sun ellipsoid)

# the tutorial uses just the x component, but we'll try to generate a 3D plot
# as well
# note that here we use a deprecated version of the kernel containing
# radii information, because the current kernel doesn't work
_, RADII_SUN = spiceypy.bodvcd(bodyid = 10, item = "RADII", maxn = 3)

radii_x = RADII_SUN[0]

# scale the position values using the Sun's x component
ssb_wrt_sun_position_scaled = ssb_wrt_sun_position / radii_x
```

[23]:
```python
# now plot the 2D trajectory of the SSB w.r.t. the Sun
ssb_wrt_sun_position_scaled_xy = ssb_wrt_sun_position_scaled[:, 0:2]

plt.style.use("dark_background")
print(plt.style.available)
```

```python
fig, ax = plt.subplots(figsize = (12, 8))

# yellow circle represents the Sun
sun = plt.Circle((0.0, 0.0), 1.0, color = "yellow")
ax.add_artist(sun)

# plot SSB movement
ax.plot(ssb_wrt_sun_position_scaled_xy[:, 0],
       ssb_wrt_sun_position_scaled_xy[:, 1],
       ls = "solid", color = "royalblue")

# format the axes
ax.set_aspect("equal")
ax.set_xlim(-2, 2)
ax.set_ylim(-2, 2)

# label
ax.set_xlabel("X Scaled to Sun-Radius")
ax.set_ylabel("Y Scaled to Sun-Radius")
ax.set_title("Most of the time, the Solar System Barycenter\n" +
            "exists outside of the Sun")
```

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background',
'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright',
'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-
darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper',
'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-
white', 'seaborn-whitegrid', 'tableau-colorblind10']
```

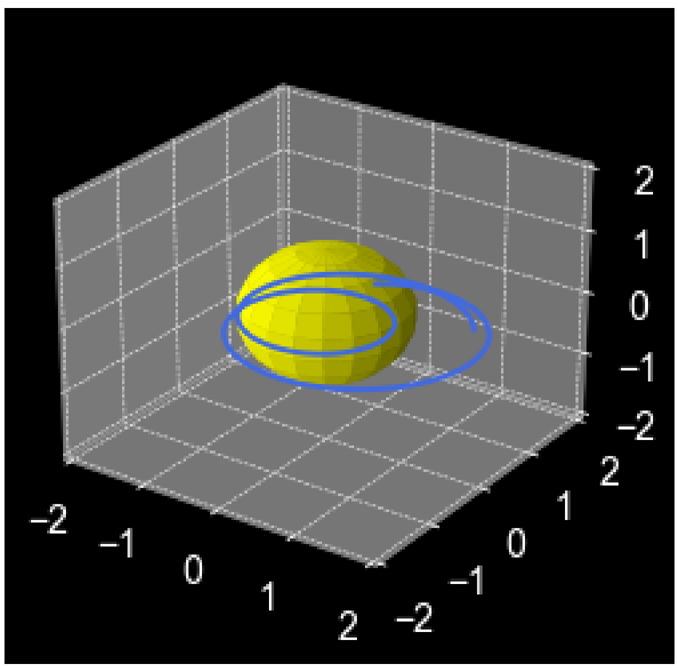
[23]:
```
Text(0.5, 1.0, 'Most of the time, the Solar System Barycenter\nexists outside of
the Sun')
```



This plot shows us that the solar system's barycenter varies greatly around the sun, despite the sun containing >99% of the solar system's mass! So technically, the sun isn't the center of the universe- it's just exists near it.

[24]:
```
# extra: plotting the 3D view
fig = plt.figure(figsize = (12, 8))


ax = fig.add_subplot(133, projection='3d')

ax.plot(ssb_wrt_sun_position_scaled[:, 0],
        ssb_wrt_sun_position_scaled[:, 1],
```

```
       ssb_wrt_sun_position_scaled[:, 2],
       ls = "solid", color = "royalblue")

# plot Sun and fill in
u, v = np.mgrid[0:2*np.pi:20j, 0:np.pi:10j]
x = np.cos(u)*np.sin(v)
y = np.sin(u)*np.sin(v)
z = np.cos(v)
ax.plot_surface(x, y, z, rstride=1, cstride=1,
                color = "yellow")

# scaling and viewing angle
ax.set_xlim(-2, 2)
ax.set_ylim(-2, 2)
ax.set_zlim(-2, 2)
ax.view_init(25, -55)
```



So this is actually an instance where the 2D top-down visualization is more informative than the 3D equivalent! Obviously space is 3D, but it's more telling to analyze these math / physics concepts in two dimensions. Lesson learned!

Our last step: calculate how long the SSB exists outside / away from the Sun.

[10]:
```
# compute the euclidian distance between the SSB and the Sun.
ssb_wrt_sun_dist_scaled = np.linalg.norm(ssb_wrt_sun_position_scaled, axis = 1)
```

```python
# compute the number of days outside of the Sun
# this is just subsetting
ssb_outside_sun_delta_days = len(np.where(ssb_wrt_sun_dist_scaled > 1)[0])

print("computation time: %s days\n" % delta_days)

print("proportion of time wehre the SSB\n lay outside the Sun: %s %%" %
      (100 * ssb_outside_sun_delta_days / delta_days))
```

```
computation time: 10000 days

proportion of time wehre the SSB
 lay outside the Sun: 64.64 %
```

This number is ~65%, which aligns visually with the length of line that exists outside the Sun.