Gabriel Rios
EAS A4170
February 25, 2022

Homework 1

1. **Download from Blackboard a file called** `calculate_elliptical_motion_sample.py`. **This is a sample Python code. Following the tutorial, run the code on Jupyter Notebook. You will see a figure. This figure shows an elliptical orbit with a circumscribed circle surrounding it. On the ellipse and circle are drawn three sets of dots in black, green and red. They correspond to the angles we discussed in class concerning elliptical orbit (true anomaly, eccentric anomaly, and mean anomaly). Study the code and answer the following questions.**

   (a) (10 points) **The symbols on the figure are represented by three quantities: *variable1*, *variable2*, and *variable3*. Which symbol corresponds to mean anomaly? Which one corresponds to eccentric anomaly? Which one corresponds to true anomaly?**

   Variable 1 corresponds to the mean anomaly. Variable 2 corresponds to eccentric anomaly. Variable 3 corresponds to true anomaly.

   (b) **(10 points) Identify these three anomalies by drawing up the angles on the figure (attach the figure in your homework).**

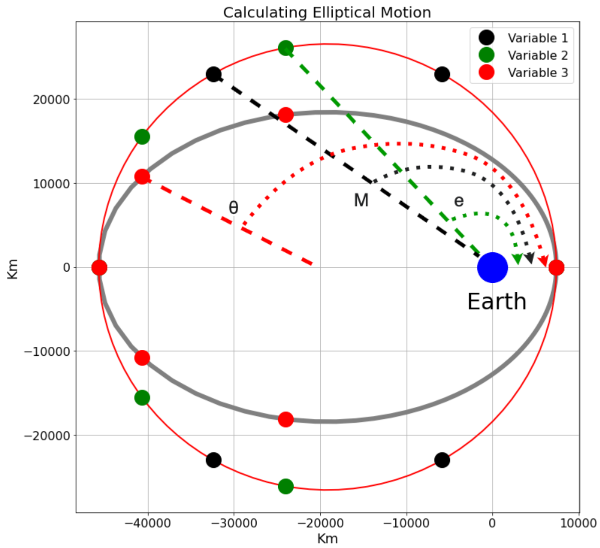


Figure 1: Satellite orbital path with quantities denoted by their symbolic names and angles from the y-axis.

   (c) **(10 points) Turn on NOAA and turn off Molniya. Rerun the code and draw two figures with eccentricity = 0.7 and 0.1, respectively. What changes do you see in satellite orbit in terms of the relation between the three anomalies? Attach your new figure.**

   When eccentricity is 0.1, the orbit is nearly circular, while when eccentricity is 0.7, the orbit becomes highly elliptical. In terms of the 3 anomalies, anomaly values are similar when the eccentricity is low as the orbit is more circular (mean and true anomaly values are almost identical at all points, rate of change of true anomaly is nearly constant), and anomaly values are very different when the eccentricity is high and the orbit is elliptical (rate of change of true anomaly is highly variable whereas rate of change of mean anomaly remains constant).

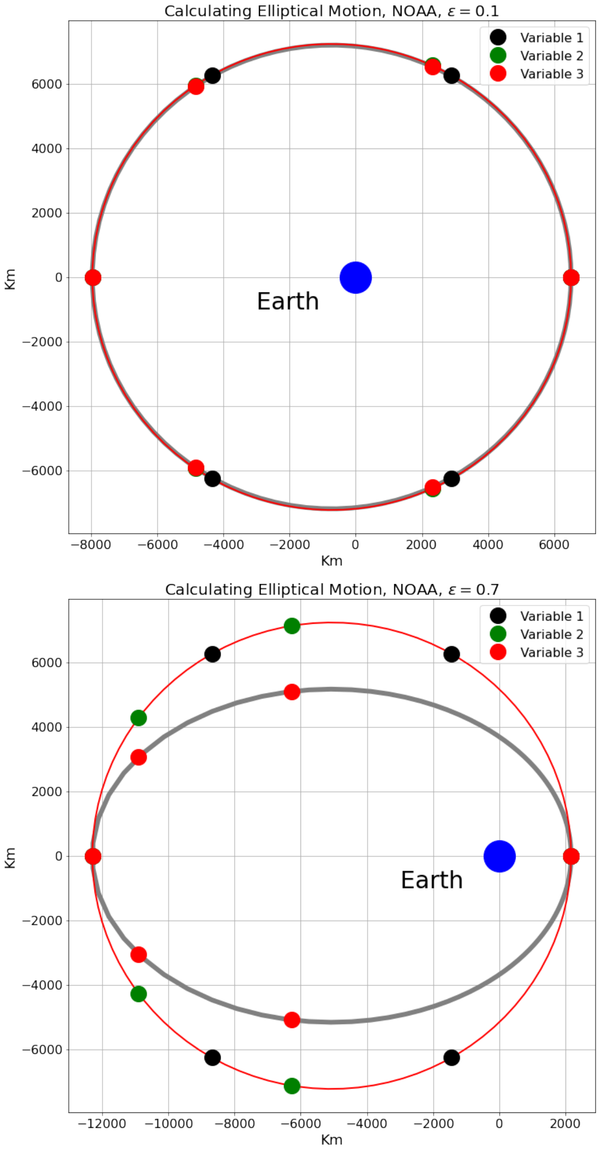


Figure 2: Plots of NOAA satellite orbital paths with $\varepsilon = 0.1$ (*left*) and $\varepsilon = 0.7$ (*right*), respectively.

2. **Download from the class website a file called** `position_satellite.py`**. The Python code is intended to calculate satellite position for a highly elliptical orbit of a weather satellite called Molniya (read Section 2.5.1 in the textbook for procedure). Study the code. Run it (with small modification) to answer the following questions:**

   (a) **(20 points) Use the code to calculate the positions of the satellite (in terms of x, y, and z, and radius, declination and right ascension) at following positions: t = 0, 1/8th period, 1/4th period, . . . , 7/8th period. Fill up the table below.**

Table 1: Satellite position properties at different stages of its orbit, as described by its period fraction.

	t = 0	t = 1/8	t = 1/4	t = 3/8	t = 1/2	t = 5/8	t = 3/4	t = 7/8
x-pos.	-1.139e+06	1.993e+07	1.954e+07	1.431e+07	7.005e+06	-1.161e+06	-9.105e+06	-1.461e+07
y-pos.	-3.128e+06	1.005e+06	9.093e+06	1.521e+07	1.921e+07	2.086e+07	1.954e+07	1.360e+07
z-pos.	-6.648e+06	1.551e+07	3.042e+07	3.833e+07	4.084e+07	3.833e+07	3.042e+07	1.550e+07
rad.	7.435e+06	2.527e+07	3.728e+07	4.366e+07	4.567e+07	4.366e+07	3.728e+07	2.527e+07
decl.	-1.106	0.660478	0.954	1.072	1.107	1.072	0.954	0.660
right asc.	-1.920	0.050	0.435467	0.816	1.221	1.626	2.007	2.392

(b) **(10 points) Now that you have 8 positions in terms of x, y and z, you can roughly draw up the orbit in a 3D space. You may use `plt.plot3d()` in Python.**

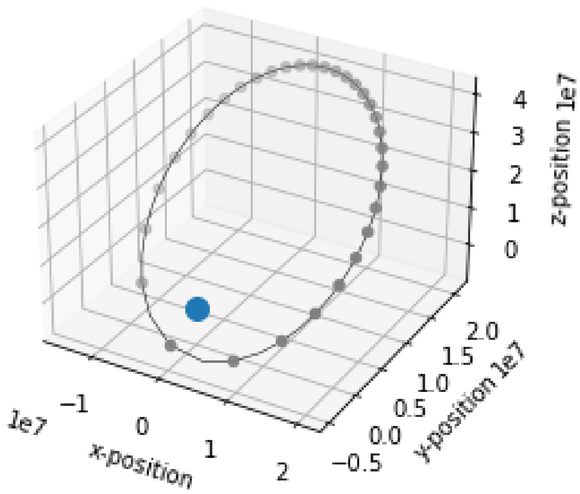


Figure 3: Molniya orbital path at multiple positions in 3D space around the Earth (blue dot).

3. **Download from the Blackboard a python file called `satellite_position_groundTrack.py`. The code is similar to `position_satellite.py` (which was used for Problem 2). But this time, instead of calculating a single position, the program calculates a whole bunch of positions using loop. Run the code and study it.**

   (a) **(10 points) Use the code to plot two complete orbits for Molniya.**

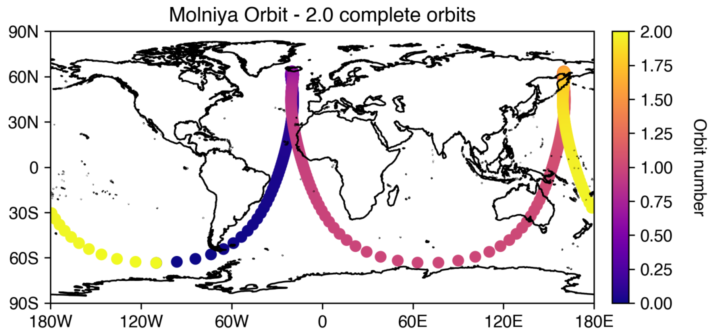
   See Figure 4a for a plot of the Molniya orbits.

   (b) **(10 points) Use the code to plot two complete orbits for NOAA polar orbiter.**

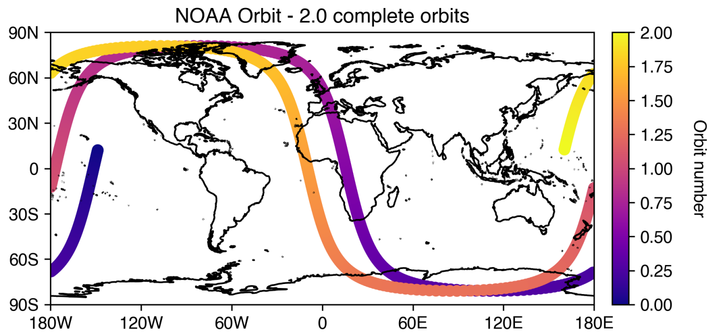
   See Figure 4b for a plot of the NOAA orbits.

   (c) **(20 points) Use the code to plot a whole day worth of orbits for the NOAA polar orbiter. *Hint*: you need to figure out how many orbits this specific satellite completes within a whole day.**

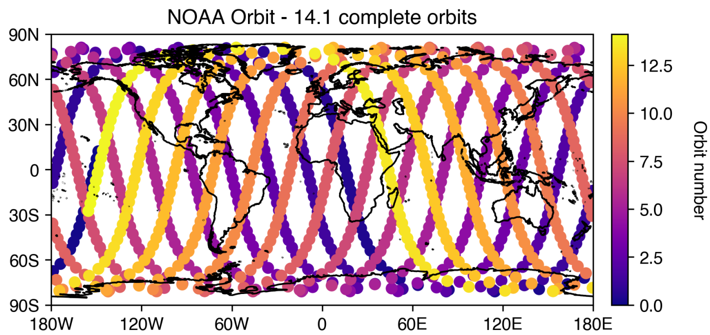
   See Figure 4c for a plot of the NOAA orbits for a full day. Note that the number of orbits (14.1) for a full day was derived by dividing the number of seconds in a day (86,400) by the NOAA polar orbiter satellite orbital period.



(a) Molniya orbital path for 2 orbits.



(b) NOAA polar orbiter orbital path for 2 orbits.



(c) NOAA polar orbiter orbital path composite for a full day (14.1 orbits).

Figure 4: Plots of various orbital paths for a pre-defined number of orbits. *Note*: the colorbar corresponds to the number of orbits completed by the satellite along its path.

EAS A4170 - Satellite Meteorology - Homework 1

File name:lcalculate_elliptical_motion_sample.py

In [ ]:

```
# Purpose: Calculate Keplerian Orbits (Elliptical Motions)
#          EAS417 (Satellite Meteorology): textbook Section 2.2
# Author: Johnny Luo
# Date: Feb 2022

# Import libraries and modules
#
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize

# Define constants
r_earth = 6.372e6 # radius of Earth
G = 6.673e-11      # gravitational const.
m_earth = 5.97e24 # mass of Earth

# Satellite parameters; 0/1: turn on/off the satellite
noaa = 1
molniya = 0

if(noaa):
    # NOAA polar orbiter
    semi_major = 7.229606e6
    epsilon = 0.7
    inclination = 98.97446 *np.pi/180
    omega_big = 29.31059*np.pi/180
    omega_small = 167.74754*np.pi/180

if(molniya):
    # Molniya orbit
    semi_major = 2.6554e7
    epsilon = 0.72
    i_angle = 63.4 *np.pi/180
    omega_big = 0*np.pi/180
    omega_small = 270*np.pi/180

# Calculate period, set up time list and three anomalies
#
T = 2*np.pi*np.sqrt((semi_major**3/G/m_earth)) # Period (Equation 2.4)
n = 2*np.pi/T # Mean motion constant (Equation 2.9)

time = np.linspace(0,T,7)  # Track 7 time stamps from 0 to one period
var1 = np.zeros(time.size) # one of the three anomalies (you need to figure out which one
var2 = np.zeros(time.size) # one of the three anomalies (you need to figure out which one
var3 = np.zeros(time.size) # one of the three anomalies (you need to figure out which one

# define a function to solve Equation 2.8
#
def equation_2pt8(eccn_anomaly, epsn, mean_anomaly):
    return eccn_anomaly - epsn*np.sin(eccn_anomaly) - mean_anomaly

# Loop over time steps to calculate the three anomalies
#
for i in range(time.size):
    var1[i] = n*time[i] # one of the three anomalies (you need to figure out which one it

    # Find the root of the Equation 2.8
    var2[i] = optimize.fsolve(equation_2pt8,0,args=(epsilon,var1[i]))
```

```
    # one of the three anomalies (you need to figure out which one it is)
    if var2[i] <= np.pi:
        var3[i] = np.arccos((np.cos(var2[i])- epsilon)/(1 - epsilon*np.cos(var2[i]))
                           )
    else:
        var3[i] = 2*np.pi -np.arccos((np.cos(var2[i])- epsilon)/(1 - epsilon*np.cos(var2[i


# Define a background ellipse to be drawn (100 points)
#
theta_1  = np.linspace(0,2*np.pi,100)
ellipse_r = semi_major*(1-epsilon**2)/(1+epsilon*np.cos(theta_1))

# The points to be drawn on the ellipse (7 points)
ellipse_r_2 = semi_major*(1-epsilon**2)/(1+epsilon*np.cos(var3))

# Making figures
#
plt.figure(figsize=(12,12))
# Plot the ellipse in gray
plt.plot(ellipse_r*np.cos(theta_1)/1000,ellipse_r*np.sin(theta_1)/1000,linewidth=6,color='
# Plot the circumscribed circle in read
plt.plot((semi_major*np.cos(theta_1)-semi_major*epsilon)/1000, semi_major*np.sin(theta_1)/
# Plot the three angles
plt.plot((semi_major*np.cos(var1)-semi_major*epsilon)/1000, semi_major*np.sin(var1)/1000,'
plt.plot((semi_major*np.cos(var2)-semi_major*epsilon)/1000, semi_major*np.sin(var2)/1000,'
plt.plot(ellipse_r_2*np.cos(var3)/1000,ellipse_r_2*np.sin(var3)/1000,'.r',markersize=40,la

# Plot the Earth at the origin
plt.legend(fontsize=16)
plt.plot(0,0,'bo',markersize=40)
plt.text(-3000,-1000,'Earth',fontsize = 30)
plt.tick_params(axis="x",labelsize=16)
plt.tick_params(axis="y",labelsize=16)
plt.xlabel('Km',fontsize=18)
plt.ylabel('Km',fontsize=18)
plt.title('Calculating Elliptical Motion, NOAA, $\epsilon = 0.7$',fontsize = 20)
plt.grid()

plt.show()
```

File name:|position_satellite.py|-|Modified|by|Gabriel|Rios

In [72]:

```
# Purpose: Calculate and plot satellite orbits
#          EAS417 (Satellite Meteorology): textbook Section 2.5.1
# Author: Johnny Luo, modified by Gabriel Rios
# Date: Feb 2022

# Load python libraries and packages
#
import numpy as np
from scipy import optimize

# Define constants
#
r_earth = 6.372e6
G = 6.673e-11
m_earth = 5.97e24

# Satellite parameters
#
noaa = 0
molniya = 1
```

```python
if(noaa):
    # NOAA polar orbiter
    semi_major = 7.229606e6
    epsilon = 0.00119958
    i_angle = 98.97446 *np.pi/180
    omega_big_0 = 29.31059*np.pi/180
    omega_small_90 = 167.74754*np.pi/180

if(molniya):
    # Molniya
    semi_major = 2.6554e7
    epsilon = 0.72
    i_angle = 63.4 *np.pi/180
    omega_big_0 = 340*np.pi/180
    omega_small_0 = 270*np.pi/180


# Calculate period and three anomalies (angles)
#
T = 2*np.pi*np.sqrt((semi_major**3/G/m_earth)) # Period (Equation 2.4)
n = 2*np.pi/T # Mean motion constant (Equation 2.9)

# Orbital perturbations (Section 2.5.1)
#
J2 = 1.08263e-3; # coefficient of the quadrupole term (Appendix E)
r_ee = 6.378137e+6; # equatorial radius of the Earth
#
# (Equations 2.12, 2.13, and 2.14)
#
dMdt = n*(1+1.5*J2*(r_ee/semi_major)**2*(1-epsilon**2)**(-1.5)*(1-1.5*(np.sin(i_angle))**2
domega_big_dt = -dMdt*(1.5*J2*(r_ee/semi_major)**2*(1-epsilon**2)**(-2)*np.cos(i_angle));
domega_small_dt = dMdt*(1.5*J2*(r_ee/semi_major)**2*(1-epsilon**2)**(-2)*(2-2.5*(np.sin(i_

# define function to calculate eccentric anamoly (Equation 2.8)
def eccentric_anomaly(e, epsn, M):
    return e - epsn*np.sin(e) - M

# Define the period interval (resolution of the path) and end period
interval, end = 1/64, 1
# Generate list of times at which calculations will be performed
times = np.arange(0, end + interval, interval)

''' Begin custom code: Gabriel Rios'''
i, arr = 0, np.full((len(times), 6), np.nan)
for time_frac in times:
    time = time_frac * T

    ''' End custom code: Gabriel Rios'''
    # Compute satellite position (following section 2.5.1)
    #

    M = n*time # Mean anomaly (increases evenly with time)
    if M>=2*np.pi: M = np.mod(M,2*np.pi)

    # Eccentric anamoly (find the root of the Equation 2.8)
    e = optimize.fsolve(eccentric_anomaly,0,args=(epsilon,M))

    # True anamaly
    if e <= np.pi:
        theta = np.arccos((np.cos(e)- epsilon)/(1 - epsilon*np.cos(e)))
    else:
        theta = 2*np.pi -np.arccos((np.cos(e)- epsilon)/(1 - epsilon*np.cos(e)))


    # Update omega_small and omega_big (Equation 2.21)
    #
```

```
    omega_small = omega_small_0 + domega_small_dt*time
    omega_big = omega_big_0 + domega_big_dt*time
    #
    # Cast into Cartesian coordinate (Equation 2.22)
    #
    radius = semi_major*(1-epsilon**2)/(1+epsilon*np.cos(theta));
    x_0 = radius*np.cos(theta);
    y_0 = radius*np.sin(theta);
    z_0 = 0;
    #
    # First rotation (Equation 2.23)
    #
    x_1 = x_0*np.cos(omega_small)-y_0*np.sin(omega_small);
    y_1 = x_0*np.sin(omega_small)+y_0*np.cos(omega_small);
    z_1 = z_0;
    #
    # Second rotation (Equation 2.24)
    #
    x_2 = x_1;
    y_2 = y_1*np.cos(i_angle)-z_1*np.sin(i_angle);
    z_2 = y_1*np.sin(i_angle)+z_1*np.cos(i_angle);
    #
    # Third rotation (Equation 2.25)
    #
    x_3 = x_2*np.cos(omega_big)-y_2*np.sin(omega_big);
    y_3 = x_2*np.sin(omega_big)+y_2*np.cos(omega_big);
    z_3 = z_2;

    # Convert the Cartesian coordinate to radius-declination-right ascension
    # (Equation 2.26)
    #
    r_s = np.sqrt((x_3**2+y_3**2+z_3**2));
    delta_s = np.arcsin(z_3/r_s);
    omega_s = np.arctan2(y_3,x_3);

    lat_s = delta_s;
    lon_s = omega_s-time*7.2921e-5; #7.2921e-5 is the rotation rate of Earth

    arr[i] = [x_3[0], y_3[0], z_3[0], r_s[0], delta_s[0], omega_s[0]]

    i += 1
```

In [ ]:

```
# 3D plotting
t, x, y, z = np.arange(0, 1, 0.125), arr.T[0, :], arr.T[1, :], arr.T[2, :]

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(0, 0, 0, s=100)
ax.scatter3D(x[::2], y[::2], z[::2], color=(0.5, 0.5, 0.5))
ax.plot3D(x, y, z, linewidth=0.5, color='k')
ax.set_xlabel('x-position')
ax.set_ylabel('y-position')
ax.set_zlabel('z-position')
```

File name: Isatellite_groundTrack.py

In [ ]:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 14 10:30:26 2022

@author: gabriel
"""
```

```python
# Purpose: Calculate and plot satellite orbits
#          EAS417 (Satellite Meteorology): textbook Section 2.5
# Author: Johnny Luo
# Date: Feb 2022

# Setting up the python environments
#
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from scipy import optimize
import cartopy.crs as ccrs

# Define constants
#
r_earth = 6.372e6
G = 6.673e-11
m_earth = 5.97e24

# Satellite parameters: 1 to turn on and 0 to turn off
#
noaa = 1
molniya = 0
satellite = 'NOAA' if noaa else 'Molniya'

if(noaa):
    # NOAA polar orbiter
    semi_major = 7.229606e6
    epsilon = 0.00119958
    i_angle = 98.97446 *np.pi/180
    omega_big_0 = 29.31059*np.pi/180
    omega_small_0 = 167.74754*np.pi/180

if(molniya):
    # Molniya
    semi_major = 2.6554e7
    epsilon = 0.72
    i_angle = 63.4 *np.pi/180
    omega_big_0 = 340*np.pi/180
    omega_small_0 = 270*np.pi/180

# Calculate period, set up time list and three anomalies
#
T = 2*np.pi*np.sqrt((semi_major**3/G/m_earth)) # Period (Equation 2.4)

print(36400/T)

n = 2*np.pi/T # Mean motion constant (Equation 2.9)

# Orbital perturbations (Section 2.5.1)
# (Equations 2.12, 2.13, and 2.14)
#
J2 = 1.08263e-3; # coefficient of the quadrupole term (Appendix E)
r_ee = 6.378137e+6; # equatorial radius of the Earth

dMdt = n*(1+1.5*J2*(r_ee/semi_major)**2*(1-epsilon**2)**(-1.5)*(1-1.5*(np.sin(i_angle))**2
domega_big_dt = -dMdt*(1.5*J2*(r_ee/semi_major)**2*(1-epsilon**2)**(-2)*np.cos(i_angle));
domega_small_dt = dMdt*(1.5*J2*(r_ee/semi_major)**2*(1-epsilon**2)**(-2)*(2-2.5*(np.sin(i_

# Set up time as a list; all parameters will be evaluated at these time stamps
#
number_of_orbits = (60*60*24/T)
time = np.linspace(0*T,number_of_orbits*T,1000) # Time increases evenly (chopped into 100(

#
```

```python
# All parameters are functions of time
#
M = np.zeros(time.size) # Mean anomaly
e = np.zeros(time.size) # Eccentric anomaly
theta = np.zeros(time.size) # True anomaly
radius = np.zeros(time.size) # radius (from Earth to satellite)
omega_small = np.zeros(time.size)
omega_big = np.zeros(time.size)

# Variables in rotation of axis
#
x_0 = np.zeros(time.size); y_0 = np.zeros(time.size); z_0 = np.zeros(time.size)
x_1 = np.zeros(time.size); y_1 = np.zeros(time.size); z_1 = np.zeros(time.size)
x_2 = np.zeros(time.size); y_2 = np.zeros(time.size); z_2 = np.zeros(time.size)
x_3 = np.zeros(time.size); y_3 = np.zeros(time.size); z_3 = np.zeros(time.size)

r_s = np.zeros(time.size); delta_s = np.zeros(time.size); omega_s = np.zeros(time.size)
lat_s = np.zeros(time.size); lon_s = np.zeros(time.size)

# define function to calculate eccentric anamoly (Equation 2.8)
def eccentric_anomaly(e, epsn, M):
    return e - epsn*np.sin(e) - M

# Loop over time steps to calculate the three anomalies
#
for i in range(time.size):

    M[i] = n*time[i] # Mean anomaly (increases evenly with time)
    if M[i]>=2*np.pi:
        M[i] = np.mod(M[i],2*np.pi)

    # Eccentric anamoly (find the root of the Equation 2.8)
    e[i] = optimize.fsolve(eccentric_anomaly,0,args=(epsilon,M[i]))
    if e[i] >= 2*np.pi:
        e[i] = np.mod(e[i],2*np.pi) # Make sure e[i] is within [0,2*np.pi]

    # True anamaly
    if e[i] <= np.pi:
        theta[i] = np.arccos((np.cos(e[i])- epsilon)/(1 - epsilon*np.cos(e[i])))
    else:
        theta[i] = 2*np.pi -np.arccos((np.cos(e[i])- epsilon)/(1 - epsilon*np.cos(e[i])))

    # Update omega_small and omega_big (Equation 2.21)
    #
    omega_small[i] = omega_small_0 + domega_small_dt*time[i]
    omega_big[i] = omega_big_0 + domega_big_dt*time[i]
    #
    # Cast into Cartesian coordinate (Equation 2.22): turn (r,theta) to (x, y, z)
    #
    radius[i] = semi_major*(1-epsilon**2)/(1+epsilon*np.cos(theta[i]));
    x_0[i] = radius[i]*np.cos(theta[i]);
    y_0[i] = radius[i]*np.sin(theta[i]);
    z_0[i] = 0;
    #
    # First rotation (Equation 2.23)
    #
    x_1[i] = x_0[i]*np.cos(omega_small[i])-y_0[i]*np.sin(omega_small[i]);
    y_1[i] = x_0[i]*np.sin(omega_small[i])+y_0[i]*np.cos(omega_small[i]);
    z_1[i] = z_0[i];
    #
    # Second rotation (Equation 2.24)
    #
    x_2[i] = x_1[i];
    y_2[i] = y_1[i]*np.cos(i_angle)-z_1[i]*np.sin(i_angle);
    z_2[i] = y_1[i]*np.sin(i_angle)+z_1[i]*np.cos(i_angle);
    #
```

```
    # Third rotation (Equation 2.25)
    #
    x_3[i] = x_2[i]*np.cos(omega_big[i])-y_2[i]*np.sin(omega_big[i]);
    y_3[i] = x_2[i]*np.sin(omega_big[i])+y_2[i]*np.cos(omega_big[i]);
    z_3[i] = z_2[i];


    # Convert the Cartesian coordinate to radius-declination-right ascension
    # (Equation 2.26)
    #
    r_s[i] = np.sqrt((x_3[i]**2+y_3[i]**2+z_3[i]**2));
    delta_s[i] = np.arcsin(z_3[i]/r_s[i]);
    omega_s[i] = np.arctan2(y_3[i],x_3[i]);

    lat_s[i] = delta_s[i];
    lon_s[i] = omega_s[i]-time[i]*7.2921e-5; #7.2921e-5 is the rotation rate of Earth

fig = plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines(resolution='50m', color='black', linewidth=1)
ax.set_extent([-180, 180, -90, 90])
xtick = np.arange(-180, 180 + 60, 60)
ytick = np.arange(-90, 90 + 30,30)
xlabel = ['180W','120W','60W','0','60E','120E','180E']
ylabel = ['90S','60S','30S','0','30N','60N','90N']
ax.set_xticks(xtick)
ax.set_yticks(ytick)
ax.set_xticklabels(xlabel)
ax.set_yticklabels(ylabel)
ax.set_title('{0} Orbit - {1:.1f} complete orbits'.format(satellite, number_of_orbits))
points = ax.scatter(lon_s*180/np.pi,
                    lat_s*180/np.pi,
                    c = time/T,
                    cmap='plasma',
                    transform = ccrs.PlateCarree())

cax = make_axes_locatable(ax)
cax_ = cax.new_horizontal(size="3%", pad=0.15, axes_class=plt.Axes)
fig.add_axes(cax_)
colorbar = fig.colorbar(points, cax=cax_)
colorbar_label = colorbar.set_label('Orbit number', rotation=270, labelpad=20)

fig.tight_layout()
```