HOMEWORK: EXERCISE 1 - Earth satellite

We consider the trajectory of an Earth spacecraft of 2000 kg mass. There is ground station located 50 deg North, 800 m above sea level, on which we define a topocentric South-East- Zenith (SEZ) reference frame S1. It is known that at 22:30 (local time at the station), the state of the spacecraft with respect to S1 is:

r1 = [5369.09, 2332.14, 656.480] km, v1 = [-3.37027, 4.90364, 0.106703] km/s.

We also define an equatorial Earth-centered inertial (ECI) reference frame S0. The station crossed the Oxz plane of S0 (with x >= 0) at 09:00 local time.

In [ ]:
```
# Data:
m_sat = 2000.0 # kg
phi = deg2rad(50.0) # rad (North)
#lambda = deg2rad(0.0) # rad (Prime Meridian)
alt = 0.8 # km
# S1: topocentric South-East-Zenith (SEZ) reference frame
# At local time = 22:30
r1 = [5369.09, 2332.14, 656.480] # km
v1 = [-3.37027, 4.90364, 0.106703] # km/s
# S0: equatorial Earth-centered inertial (ECI) reference frame
# At local time = 9:00 crosses Oxz plane (x >=0)
```

```
3-element Vector{Float64}:
 -3.37027
  4.90364
  0.106703
```

To use the AstrodynamicsEdu.jl package, first we need to activate the new environment and add the package from its GitHub repository

In [ ]:
```
using Pkg
# Pkg.activate(".")
Pkg.activate("C:\\Users\\alici\\Desktop\\Julia\\HOMEWORK")
Pkg.instantiate()
Pkg.add(url="https://github.com/AliciaSBa/AstrodynamicsEdu.jl.git")

using AstrodynamicsEdu
using LinearAlgebra
```

```
  Activating project at `C:\Users\alici\Desktop\Julia\HOMEWORK`
    Updating git-repo `https://github.com/AliciaSBa/AstrodynamicsEdu.jl.git`
    Updating registry at `C:\Users\alici\.julia\registries\General.toml`
   Resolving package versions...
  No Changes to `C:\Users\alici\Desktop\Julia\HOMEWORK\Project.toml`
  No Changes to `C:\Users\alici\Desktop\Julia\HOMEWORK\Manifest.toml`
```

(a) Implement a function `[r0,v0] = SEZ2ECI(r1,v1,rS,phi,t)` that computes the position and velocity vectors r0, v0 in ECI (expressed in the ECI basis), provided the position and velocity vectors r1, v1 in SEZ (expressed in the SEZ basis), the radius rS of the station (in km) and latitude ϕ (in rad) of the observer from the center of the Earth, and the time t in hours since the observer crossed the Oxz plane of ECI. Use units of km and km/s. Apply it to compute r0, v0 using the initial spacecraft data and report your results.

The r1 and v1 vectors define the position and velocity of the Point of the satellite with respect to reference frame S1 (i.e. SEZ). We need to find its coordinates with respect to the CanonicalReferenceFrame (i.e. ECI). A point expressed with respect to the CanonicalReferenceFrame is simply a MyPoint.

To create a MyPoint, first we need to define the reference frame S1 as a MyReferenceFrame. For that we need to obtain Basis 1, which is the CanonicalBasis rotated by $\theta = \omega_E t + \Lambda$ wrt z-axis, where $\Lambda = 0$ and ω_E is the rotational speed of the Earth; and then by $-\phi$ wrt y-axis. It is important to remember that as the Earth is rotating, there is an angular velocity of $\omega_E k_0$ expressed in the CanonicalBasis. We also need to obtain the origin point of S1, which is the the station. Considering relative kinematics,

$$\mathbf{r_0^{Station}} = [{}_0R_1] * [0, 0, r_{Station}]$$

$$\mathbf{v_0^{Station}} = \omega_{10} \times \mathbf{r_0^{Station}}$$

Knowing this, we would just need to use the constructor of MyPoint for a point defined in a non-canonical reference frame.

In [ ]:
```
function SEZ2ECI(r1::Vector,v1::Vector,rStation::Float64,phi::Float64,t0::Float64)
    # Calculate the siderial time
    omega_Earth = 2*pi/86400 # rad/s
    theta = omega_Earth*t0 # rad

    # Obtain the basis of S1 wrt S0
    R_0_1 = [cos(theta)*sin(phi) -sin(theta) cos(theta)*cos(phi);
                sin(theta)*sin(phi) cos(theta) sin(theta)*cos(phi);
                -cos(phi) 0.0 sin(phi)]
    i1 = MyVector(R_0_1*i0.c0)
    j1 = MyVector(R_0_1*j0.c0)
    k1 = MyVector(R_0_1*k0.c0)
    omega_b1_0 = MyVector([0.0,0.0,omega_Earth])
    alpha_b1 = MyVector([0.0,0.0,0.0])
    basis1 = MyBasis(i1, j1, k1, omega_b1_0, alpha_b1)

    # Obtain the origin point of the station wrt S0
    rStation0 = MyVector([rStation*cos(phi)*cos(theta), rStation*cos(phi)*sin(theta),
                                                        rStation*sin(phi)])
    println("rStation0 = ", rStation0)
    vStation0 = cross(omega_b1_0,rStation0)
    println("vStation0 = ", vStation0)
    aStation0 = MyVector([0.0,0.0,0.0])
    Station = MyPoint(rStation0, vStation0, aStation0)

    # Obtain the referece frame S1, defined by the basis1 and the origin point of Station
    S1 = MyReferenceFrame(basis1, Station)

    # Obtain the position and velocity of the satellite wrt S0
    Sat0 = MyPoint(r1, v1, [0.0,0.0,0.0], S1)
    r0 = Sat0.position
    v0 = Sat0.velocity

    return r0, v0
end

rStation = R_Earth + alt # km
t0 = (22.5 - 9.0)*3600.0 # s
r0,v0 = SEZ2ECI(r1,v1,rStation,phi,t0)
```

```
println("r0 = ", r0, " km")
println("v0 = ", v0, " km/s")
```

```
rStation0 = MyVector([-3783.946420063161, -1567.3619264832814, 4881.081982665504])
vStation0 = MyVector([0.11398177578139404, -0.27517634895481075, 0.0])
r0 = MyVector([-7081.212190082387, -5457.424273163714, 1932.7903113923494]) km
v0 = MyVector([4.595300978204476, -4.083577028931066, 2.2481070375123773]) km/s
```

(b) Compute the classical orbital elements a, e,Ω, i, ω of the spacecraft, and the value of the true anomaly θ0 at the instant of the observation. Plot the orbit.

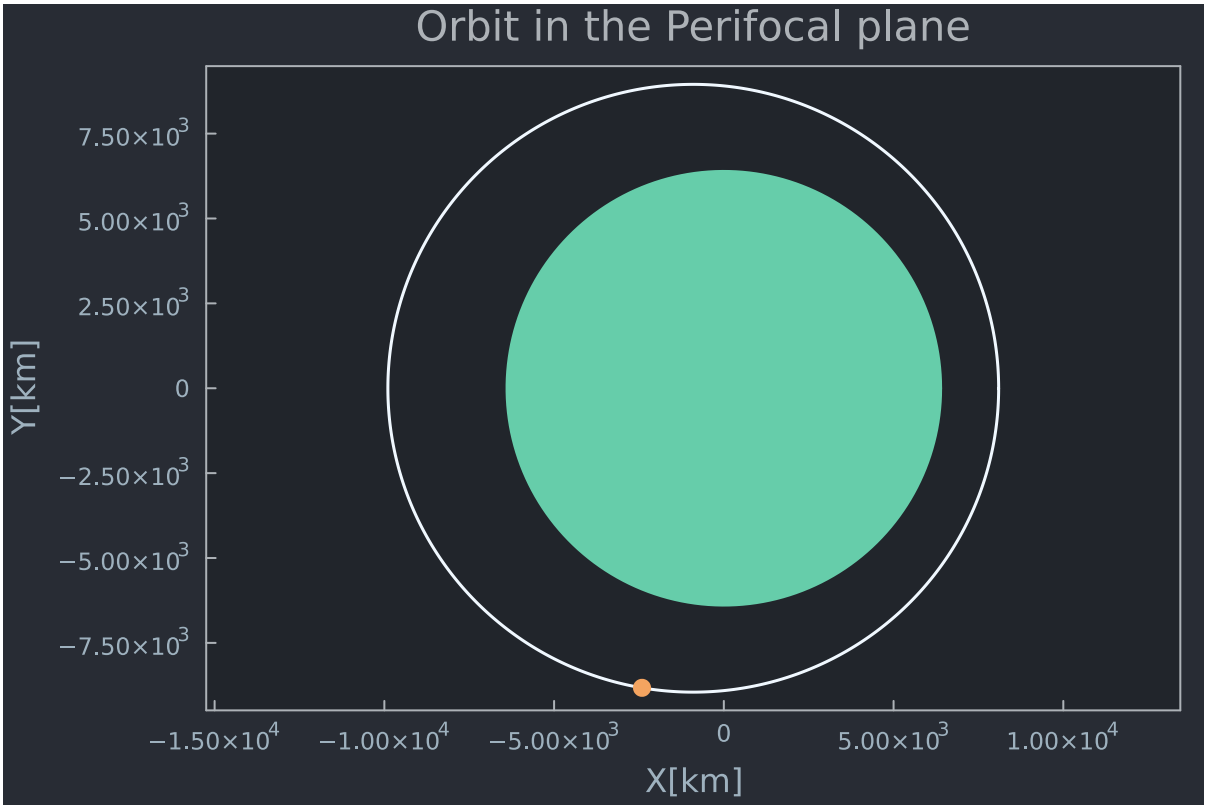
To use the `stateVector_to_COE` defined first we need to create a MyStateVector object with the position and velocity MyVector objects already computed, and then introduced the constant μ of the Earth, defined in the package.

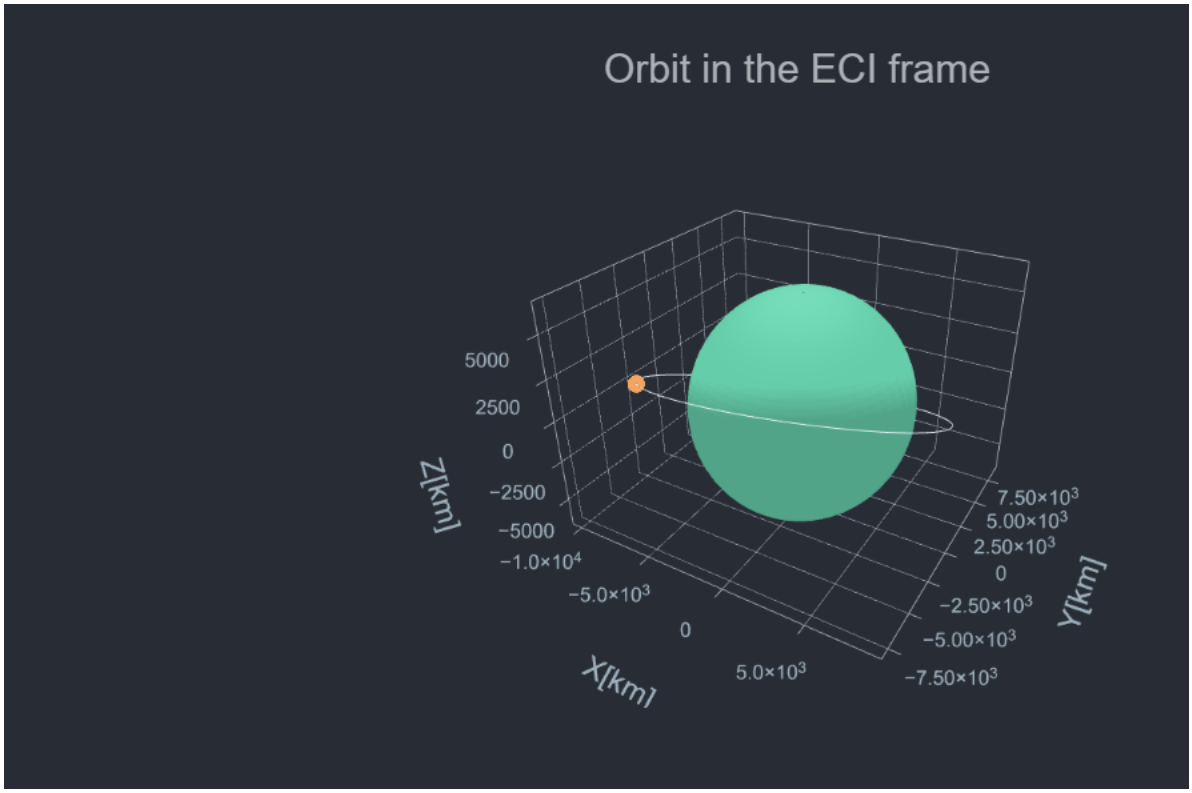
In [ ]:
```
state0 = MyStateVector(r0, v0)
coe = stateVector_to_COE(state0, mu_Earth)
println("a = ", coe.a, " km")
println("e = ", coe.e)
println("inc = ", coe.inc, " rad")
println("RAAN = ", coe.RAAN, " rad")
println("omega = ", coe.omega, " rad")
println("theta0 = ", coe.theta, " rad")
```

```
a = 8995.61613586091 km
e = 0.10010587914623008
inc = 0.4364211510849171 rad
RAAN = 3.316245899179842 rad
omega = 2.36092534167557 rad
theta0 = 4.445749406362666 rad
```

We can plot the orbit both in the perifocal plane or in the ECI reference frame using the built-in functions of `plot_orbit_perifocal` or `plot_orbit_ECI` respectively.

In [ ]:
```
#Pkg.add("PlotlyJS"); Pkg.add("PlotlyBase")
#using PlotlyJS, PlotlyBase
plot1 = plot_orbit_perifocal(coe, R_Earth)
display(plot1)
plot2 = plot_orbit_ECI(coe, R_Earth, mu_Earth)
display(plot2)
```





With the r0 computed in part a, we can find that the orbit has an altitude of around 2,775 km, so it is at the beginning of the Medium Earth Orbit (MEO), which corresponds to an altitude from 2,000 km to 35,786 km above sea level. As it was expected from the Classical Orbital Elements obtained, we have a near circular orbit with an inclination of around 22º.

(c) Determine the local time at the station when the spacecraft would reach θ = θ0 + π/2. Determine also the position and velocity vectors of the spacecraft in ECI 6 h after the initial observation. Finally, implement a function `[r1,v1] = ECI2SEZ(r0,v0,rS,phi,t)` that does the inverse of SEZ2ECI, and compute the position and velocity vectors of the spacecraft in SEZ at that instant.

To find the local time we would need to add the time of flight Δt to the initial time the position was measured (22:30). For that we would just need to use the `timeOfFlight` function from the package.

In [ ]:
```
## Determine the local time at the station when the spacecraft would reach ϑ = ϑ0 + π/2.
theta2 = coe.theta + pi/2
coe2 = MyCOE(RAAN = coe.RAAN, inc = coe.inc, omega = coe.omega, a = coe.a, e = coe.e,
                                                                    theta = theta2)
Dtime = timeOfFlight(coe,coe2,mu_Earth)
# Dtime = timeSinceTrueAnomaly(state0,mu_Earth,coe.theta,theta2)
println("Dtime = ", Dtime/3600, " h")

localTime = 22.5 + Dtime/3600.0 # h
Hour = floor(localTime)
Minutes = floor((localTime - Hour)*60.0)
Seconds = round(((localTime - Hour)*60.0 - Minutes)*60.0,digits=2)
println("Local Time = ", Hour, " h  ", Minutes, " min  ", Seconds, " s  ")
```

```
Dtime = 0.5342585085344397 h
Local Time = 23.0 h  2.0 min  3.33 s
```

To determine the position and velocity vectors of the spacecraft in ECI 6 h after the initial observation we would have to make use of Kepler's equations to go from the time to the trueAnomaly. And then convert from the COE to the state vector. Thankfully for us, these functions already exist.

In [ ]:
```
# First, let's calculate the time since periapsis for the initial observation
time0 = timeSincePeriapsis(state0,mu_Earth,coe.theta) # s
println("time0 = ", time0, " s")
# Now, let's calculate the time since periapsis for the observation 6 h later
time3 = time0 + 6*3600.0 # s
# So we can obtain that for this time, the true anomaly is
theta3 = timeSincePeriapsis_to_trueAnomaly(state0,mu_Earth,time3)
coe3 = MyCOE(RAAN = coe.RAAN, inc = coe.inc, omega = coe.omega, a = coe.a, e = coe.e,
                                                                    theta = theta3)
println("theta3 = ", coe3.theta, " rad")
# Let's convert the COE to state vector in ECI frame at this time
state3 = COE_to_stateVector(coe3,mu_Earth)
r3_0 = state3.r
v3_0 = state3.v
println("r3_0 = ", r3_0, " km")
println("v3_0 = ", v3_0, " km/s")
```

```
time0 = 6273.735313339321 s
theta3 = 1.966523657851537 rad
r3_0 = MyVector([2073.3694816878774, 8266.427689909542, -3628.895472057815]) km
v3_0 = MyVector([-6.112701337049588, 1.6711825727263003, -1.2630271576367642]) km/s
```

To convert from ECI to SEZ, we would have to follow a similar approach to before, now the only difference is that as we want the components in the S1 reference frame, we would need to use the `pos_vel_acc_inRF` function.

In [ ]:
```
# Function to convert from ECI to SEZ
function ECI2SEZ(r0::MyVector,v0::MyVector,rStation::Float64,phi::Float64,t0::Float64)

    # Calculate the siderial time
    omega_Earth = 2*pi/86400 # rad/s
    theta = omega_Earth*t0 # rad

    # Obtain the basis of S1 wrt S0
    R_0_1 = [cos(theta)*sin(phi) -sin(theta) cos(theta)*cos(phi);
             sin(theta)*sin(phi) cos(theta) sin(theta)*cos(phi);
             -cos(phi) 0.0 sin(phi)]
    i1 = MyVector(R_0_1*i0.c0)
    j1 = MyVector(R_0_1*j0.c0)
    k1 = MyVector(R_0_1*k0.c0)
```

```
    omega_b1_0 = MyVector([0.0,0.0,omega_Earth])
    alpha_b1 = MyVector([0.0,0.0,0.0])
    basis1 = MyBasis(i1, j1, k1, omega_b1_0, alpha_b1)

    # Obtain the origin point of the station wrt S0
    rStation0 = MyVector([rStation*cos(phi)*cos(theta), rStation*cos(phi)*sin(theta),
                                                        rStation*sin(phi)])
    vStation0 = cross(omega_b1_0,rStation0)
    aStation0 = MyVector([0.0,0.0,0.0])
    Station = MyPoint(rStation0, vStation0, aStation0)

    # Obtain the referece frame S1, defined by the basis1 and the origin point of Station
    S1 = MyReferenceFrame(basis1, Station)

    # Obtain the position and velocity of the satellite wrt S1
    Sat0 = MyPoint(r0, v0, MyVector([0.0,0.0,0.0]))
    r1,v1,a1 = pos_vel_acc_inRF(Sat0,S1)

    return r1, v1
end

## Compute the position and velocity vectors of the spacecraft in SEZ at that instant,
        # using the function ECI2SEZ.
t3 = t0 + 6*3600.0 # s
r3_1, v3_1 = ECI2SEZ(r3_0,v3_0,rStation,phi,t3)
println("r3_S1 = ", r3_1, " km")
println("v3_S1 = ", v3_1, " km/s")
```

```
r3_S1 = [-2909.9993379362068, 5078.968549237365, -13550.765972637573] km
v3_S1 = [-1.8799054401191884, -4.510175011507853, -3.226192819655835] km/s
```

(d) At exactly 6 h from the first observation, our spacecraft releases a secondary payload it carried within, of 500 kg, with a velocity v = [-6.0, 3,-1.6] km/s in ECI. Determine the classical orbital elements of the spacecraft after this release operation.

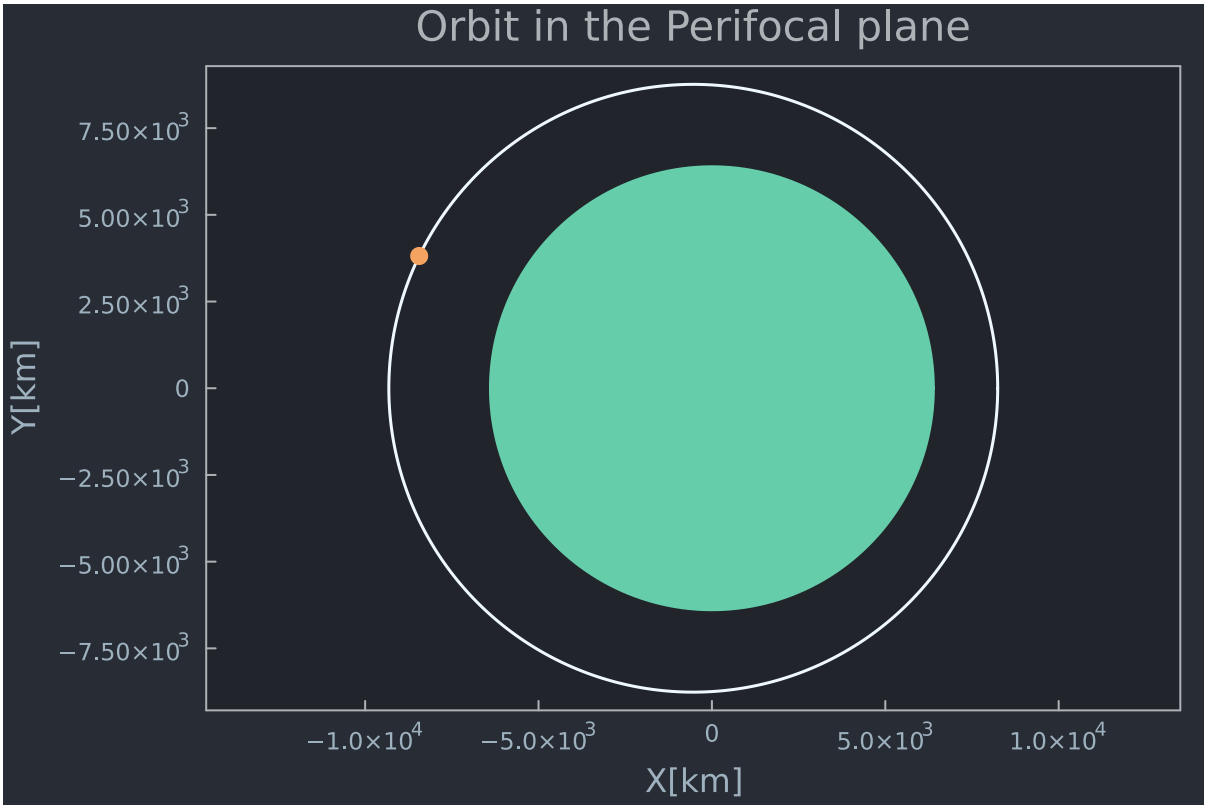
The position at that instant would be the same, however, because of the change in mass, the velocity of the aircraft would change. To obtain it we just need to apply conservation linear momentum: $p = m * v$.

In [ ]:
```
# Set the new position and speed of the spacecraft after the satellite is deployed
r4_0 = r3_0 # km

# Obtain the speed from the conservation of linear momentum (p = m*v),
# consider that the mass of the spacecraft changes from 2000 kg to 1500 kg
m_sc = 2000.0 # kg
m_sat = 500.0 # kg
m_new = m_sc - m_sat # kg
v_sat = MyVector([-6.0, 3.0,-1.6]) # km/s
v4_0 = (m_sc*v3_0 - m_sat*v_sat)/m_new # km/s

# Use the new state vector to obtain the new COE
state4 = MyStateVector(r4_0, v4_0)
coe4 = stateVector_to_COE(state4,mu_Earth)
println("RAAN = ", coe4.RAAN, " rad")
println("inc = ", coe4.inc, " rad")
println("omega = ", coe4.omega, " rad")
println("a = ", coe4.a, " km")
println("e = ", coe4.e)
println("theta = ", coe4.theta, " rad")

# Plot the new orbit
plot3 = plot_orbit_perifocal(coe4,R_Earth)
display(plot3)
```



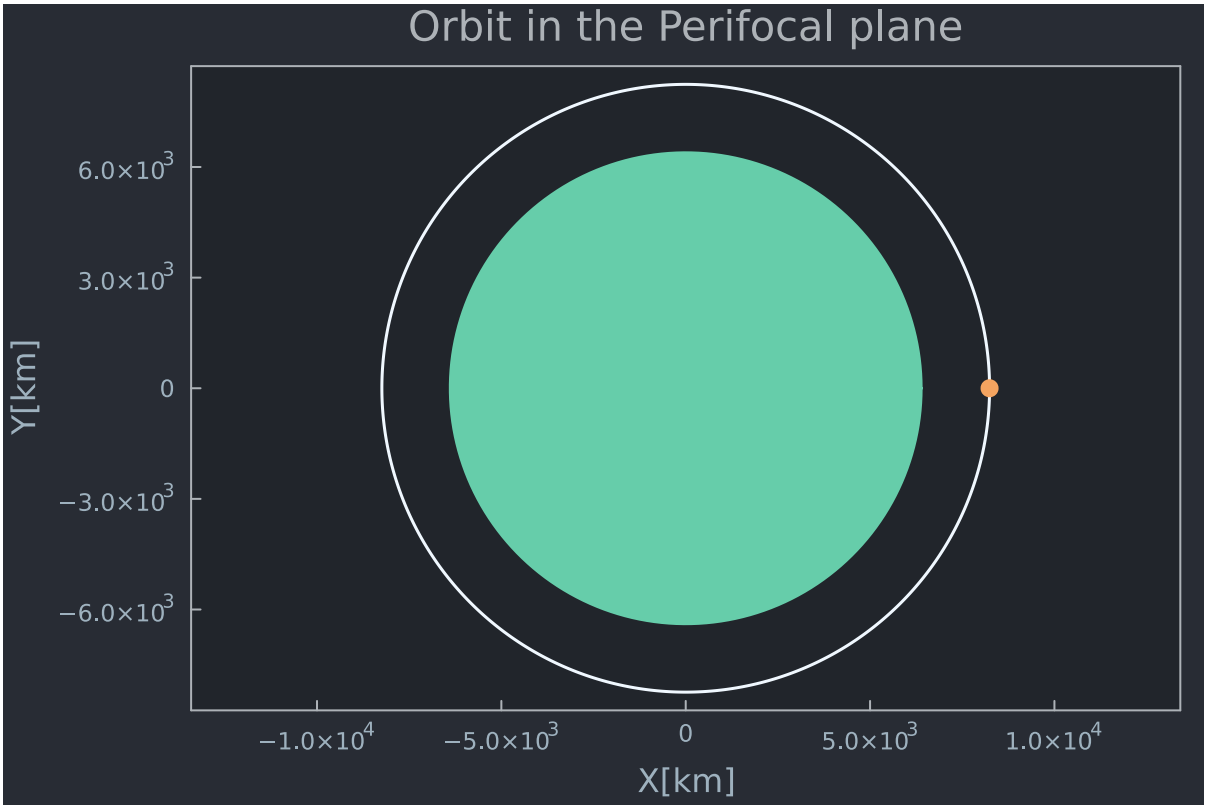

```
RAAN = 3.343426453001852 rad
inc = 0.4412667299383676 rad
omega = 1.5855775112078734 rad
a = 8778.734728431185 km
e = 0.06126011793632399
theta = 2.717266123784746 rad
```

(e) Compute the ΔV required to circularize the orbit of the spacecraft at its new pericenter.

The ΔV needed would just be the difference between the circular velocity of the new orbit and velocity at the periapsis, which can be calculated using the `speed_visViva` function.

In [ ]:
```
# First we need to obtain the rp4 of the last orbit computed
rp4 = coe4.a*(1.0 - coe4.e) # km
# The radius of the circular orbit is the same as the rp4
R_circ = rp4 # km
e_circ = 0.0
# Calculate the circular velocity and the speed of the spacecraft in the last orbit
v_circ = circularVelocity(R_circ,mu_Earth)
vp4 = speed_visViva(rp4,coe4.a,mu_Earth)
# Calculate the DeltaV
DeltaV = abs(v_circ - vp4) # km/s
println("DeltaV = ", DeltaV, " km/s")

# Validate the orbit by plotting it
coeCirc = MyCOE(RAAN=coe4.RAAN, inc=coe4.inc, omega=coe4.omega, a=R_circ, e=e_circ,
                                                                    theta=0.0)
plot4 = plot_orbit_perifocal(coeCirc,R_Earth)
display(plot4)
```

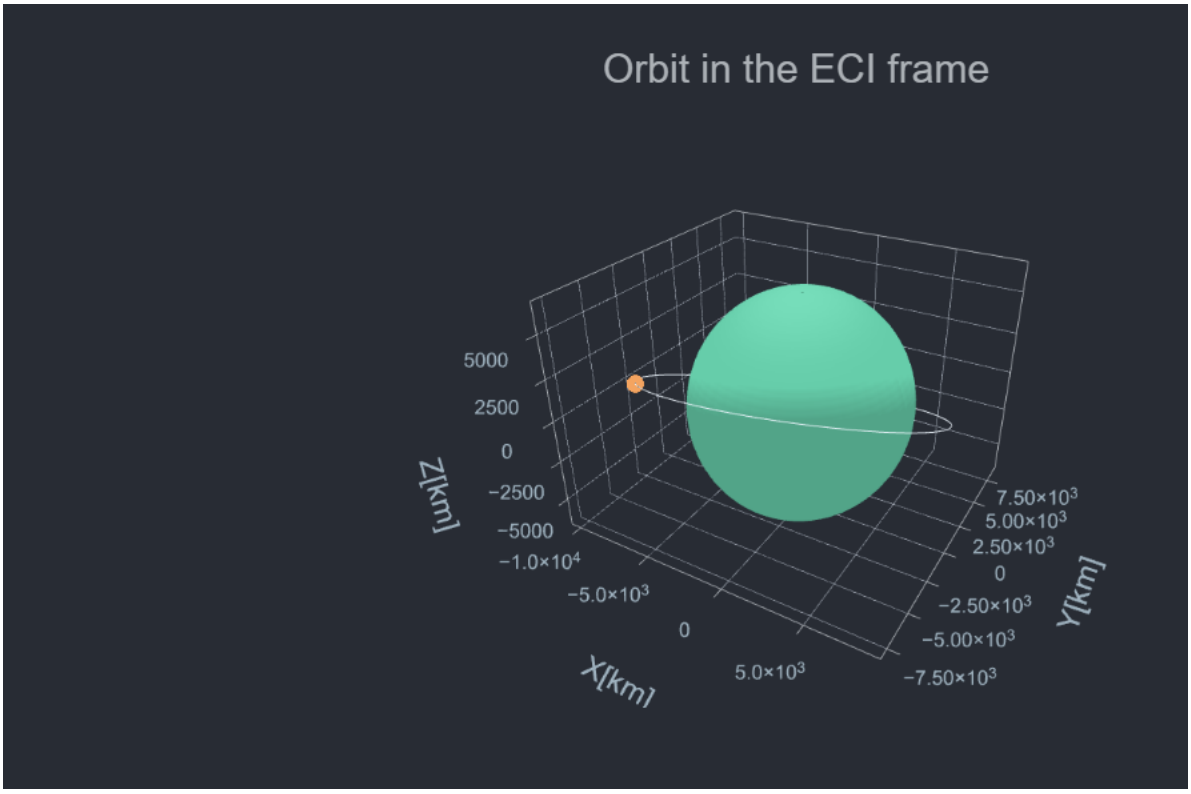


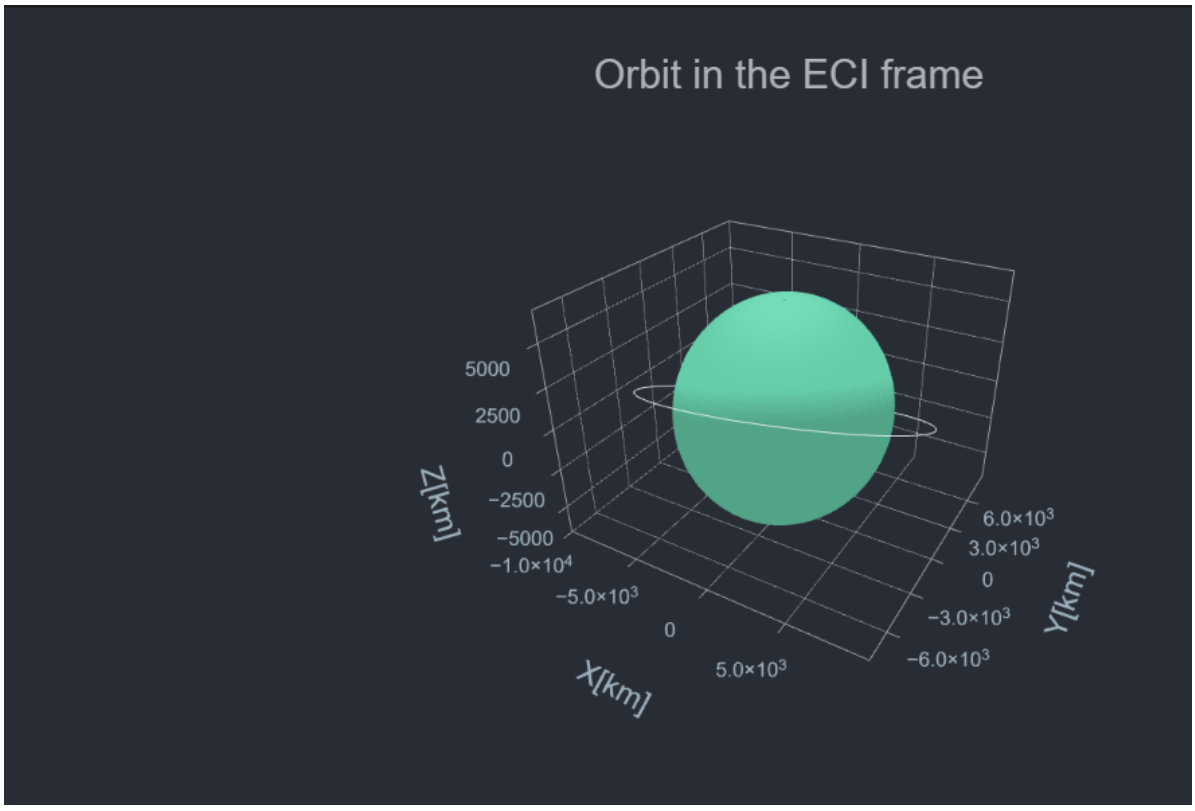

```
DeltaV = 0.20985759248770997 km/s
```

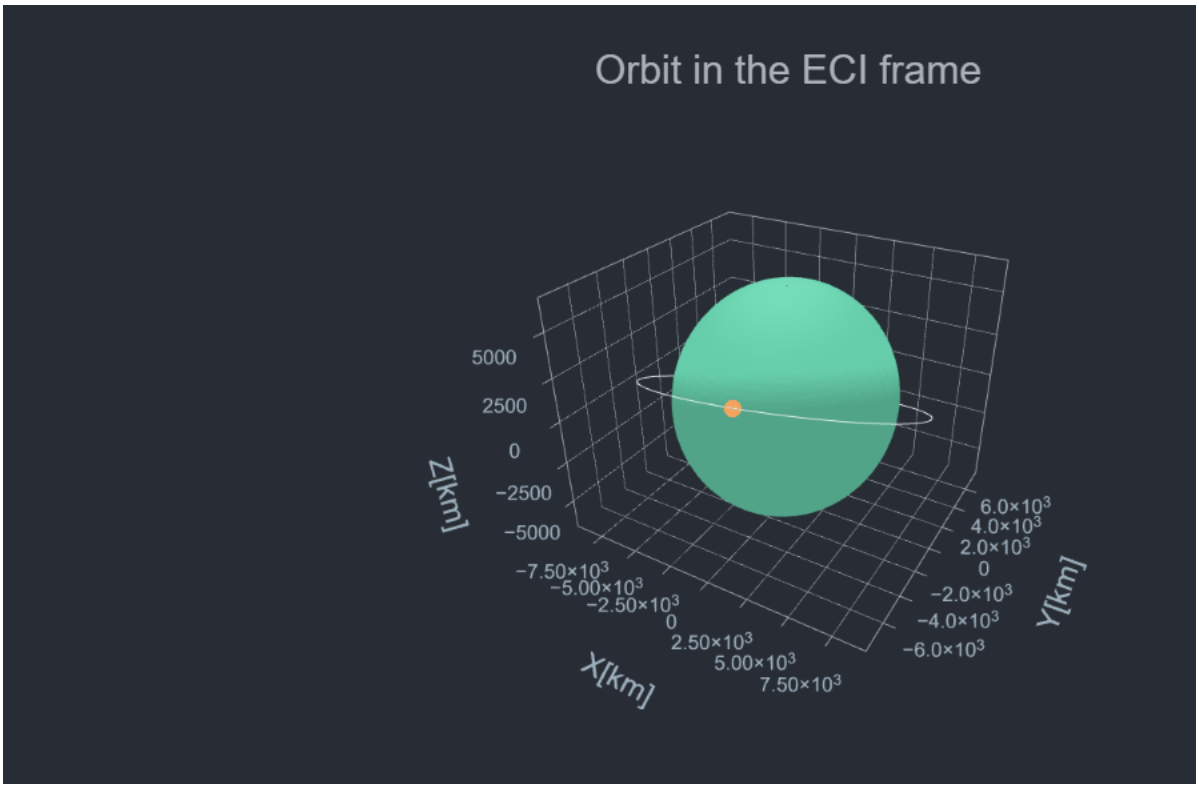
As expected the ΔV needed is quite small as the eccentricity of the previous orbit was very close already to zero.

Now, to further visualize the problem, let's plot in 3D all the different orbits obtained. Using the `plotly()` backend we can rotate and move around the figure to see the orbit from different perspectives.

In [ ]:
```
####### PLOTTING ALL THE ORBITS IN 3D ########
#using PlotlyJS, PlotlyBase
plotInitialOrbit_ECI = plot_orbit_ECI(coe, R_Earth, mu_Earth)
display(plotInitialOrbit_ECI)
plotAfterEjection_ECI = plot_orbit_ECI(coe4, R_Earth, mu_Earth)
display(plotAfterEjection_ECI)
plotCircular_ECI = plot_orbit_ECI(coeCirc, R_Earth, mu_Earth)
display(plotCircular_ECI)
```


Orbit in the ECI frame
5000
2500
0
−2500
−5000
Z[km]
−1.0×10^4
−5.0×10^3
0
5.0×10^3
X[km]
7.50×10^3
5.00×10^3
2.50×10^3
0
−2.50×10^3
−5.00×10^3
−7.50×10^3
Y[km]


Orbit in the ECI frame
5000
2500
0
−2500
−5000
Z[km]
−1.0×10^4
−5.0×10^3
0
5.0×10^3
X[km]
6.0×10^3
3.0×10^3
0
−3.0×10^3
−6.0×10^3
Y[km]


Orbit in the ECI frame
5000
2500
0
−2500
−5000
Z[km]
−7.50×10^3
−5.00×10^3
−2.50×10^3
0
2.50×10^3
5.00×10^3
7.50×10^3
X[km]
6.0×10^3
4.0×10^3
2.0×10^3
0
−2.0×10^3
−4.0×10^3
−6.0×10^3
Y[km]