Python Codes of Visible Passes

There are fifteen codes that was written by Tahsin Çağrı Şişman. Fourteen codes are converted by İbrahim Selçuk Öztürk to Python language. So codes could not finish yet. The problem that I faced during writing, could not be worked the codes together at the same time. The main code that is Visible Passes is containing all the other fourteen codes names. So this is unfinished one. But I did another attempt that is collecting all the codes inside to the one code that I named it main code. And this is also could not work properly. The python codes are given below. And in the end there other attempt code as a second way. Visible_Passes function is the main function finding the visible passes. This code is written by Tahsin Çağrı Şişman and closely related with the code developed for the paper "Uyduların Belirli Bir Bölgeden Görünür Geçişlerinin Saptanması" by Mehmet Fatih Ertürk, Şahin Ulaş Köprücü, Uzay Tuğcular, İbrahim Arda, Yılmaz Barış Erkan, and Tahsin Çağrı Şişman. In writing this code, for some parts M-files provided by Howard Curtis are directly used (sv_from_coe.m, LST.m, los.m) or modified (solar_position.m). These uses are cited at the related points. And some of them mentioned in Turkish.

1. First Way of Visible Passes Code

TLE_to_COE_and_t0

TLE_to_COE_and_t0 one of the most important codes among these codes. Because this is the input code which takes Two Line Element (TLE) data from data source which is taken from space-track.org and puts inside of the code. The main reason of the putting TLE data to the code is that TLE data gives information about satellite. For example, the time thats passed from the launch of satellite to present day or the six parameters which they are classic orbital elements, the key terms for orbital analysis.

```
import numpy
import math

def TLE_to_COE_and_t0():

    #TLE_to_COE_and_t0 fonksiyonu TLE verisini içeren TLE dosyasını
okur ve
    #bu TLE verilerini klasik yörünge elemanlarına çevirir, başlangıç
zamanı da.

    #Fonksiyonun çıktıları klasik yörünge elemanları (özgül açısal
momentum,
    #eğiklik, yükselme düğümü açısı, basıklık, perigee argümanı, doğru
açıklık),
    #TLE çağının yılı, TLE çağı için gün, gün fraksiyonu ile, ortalama
açıklık ve
    #ortalama hareket

    #Output için, tüm açısal birimler radyan cinsindendir.

    #kye0              -TLE datanın klasik yörüge elemanları
[h,i,Omega,e,omega,theta]
    #year              -TLE çağının yılı
    #day               -TLE çağı için gün, fraksiyonu ile birlikte
    #Me                -TLE çağı için Ortalama açıklık
```

```
    #n                 -TLE çağı için Ortalama hareket
    #Observation_Site -Doğu boylamı içeren, gözlem noktasının enlemi ve
yüksekliği
    #date              -Anlık tarih
    #UT                -Anlık evrensel zaman (Universal time)
    #EL                -Gözlem noktasının Doğu boylamı (derece)
    #Lat               -Gözlem noktasının enlemi (derece)
    #lstOS             -Gözlem noktasının yıldız zamanı (derece)
    #H                 -Gözlem noktasının yüksekliği (km)

    #TLE dosyası tle.txt okunması. Uygun bir tle.text dosyası
oluşturmak için,
    #space-track.org dan TLE verisi elde edin ve kopyalayın ve tle.text
    #dosyasına yapıştırın.
    #Aşağıdaki bölümde, TLE verisinin ilk satırı Line 1 cell array
olarak okunur ve
    #TLE verisinin ikinci satırı Line 2 cell array olarak okunur.

    fileID = open('tle.txt', 'r')
    fileID.seek(18)
    epoch = float(fileID.read(5)
    fileID.seek(40)
    n0dot = 2.0*(float(fileID.read(3))/1000000)
    fileID.seek(45)
    n0doubledot1 = 6.0*(float(fileID.read(1)))
    fileID.seek(51)
    n0doubledot2 = 10**float(fileID.read(1))
    n0doubledot = n0doubledot1*n0doubledot2
    fileID.seek(54)
    Bstar1 = float(fileID.read(5))
    fileID.seek(60)
    Bstar2 = 10**(-float(fileID.read(1)))
    Bstar = Bstar1*Bstar2
    fileID.seek(80)
    i = float(fileID.read(7))*math.pi/180.0
    fileID.seek(88)
    Omega = (float(fileID.read(8)))*math.pi/180.0
    fileID.seek(100)
    e = (float(fileID.read(4)))/(10**7)
    fileID.seek(105)
    omega = (float(fileID.read(8)))*math.pi/180.0
    fileID.seek(114)
    Me = (float(fileID.read(8)))*math.pi/180.0
    fileID.seek(123)
    # Ortalama hareket, rad/s biriminde
    n = (float(fileID.read(7)))*2.0*math.pi/(24.0*3600.0)

    #Klasik yörünge elemanlarının bulunuşu, bunlar; h, i, Omega, e,
omega ,theta
    #TLE data çıktıları. i, Omega, e, omega TLE dosyasından okunabilir.
h ve theta,
    #n ve Me'ye göre bulunur.

    #Kütleçekimsel parametre mu, km^3/s^2:
    mu = 398600

    #h'ın n'den elde edilişi:
    h = (mu**2/n)**(1/3)*math.sqrt(1-e**2)
```

```
    #Kepler denklemini çözerek theta'nın Me'den elde edilişi:

    def f(E):
            KeplerEqn = E - e*math.sin(E) - Me
            return KeplerEqn
    #Kepler denkleminin çözümü, başlangıç tahmini Pi.
            E = fsolve(f,[math.pi])

            if math.tan(E/2.0)<0:
                theta = 2.0*math.atan(math.sqrt((1+e)/(1-
e))*math.tan(E/2.0)) + 2.0*math.pi;
            else:
                theta = 2.0*math.atan(math.sqrt((1+e)/(1-
e))*math.tan(E/2.0));
                #Klasik yörünge elemanları(kye). Ortalama hareketin
birimi,
                #rad/s ve bütün açılar radyan cinsinden.
                kye = numpy.array([h, i, Omega, e, omega, theta])

    #Çağ, Evrensel Zamanı tanımlar (Greenwich'teki güneş zamanı).
Aşağıda,
    #çağ yıla çevrilmiştir, yılın günü gün kesri ile

    year = 2000 + (epoch - numpy.mod(epoch,1000))/1000

    day = numpy.mod(epoch,1000)



TLE_to_COE_and_t0()
```

1.1.1 Output

Output code is another important code, because it gives output data which is visible pass date, maximum altitude, time, elevation, azimuth. It writes output in text file. The code is given below:

```
import sys
import numpy


def Output(i, fileID, date, data):

    #Output fonksiyonu, görünür geçiş çıktısını text dosyasına yazar.

    #i      - Output(çıktı) tipi; start(başlangıç), maximum
altitude(maksimum yükseklik), end(son)
    #fileID -Çıktının yazıldığı dosyanın ismi
    #date   -Geçişin tarih ve zamanı
    #data   -Geçişin verisi

    i=1
```

```
    fileID = open('Visible_pass_output.txt','w+')
    if i == 1:
        fileID.write('Date of visible pass: {} {} {} \n'
.format(round(date[3]), round(date[2]), round(date[1])))
        fileID.write('\n')
        fileID.write('Start of the visible pass:\n ')
    elif i == 2:
            fileID.write('Maximum altitude of the visible pass:\n')
    elif i == 3:
                fileID.write('End of the visible pass:\n')

    fileID.write('Time:  {} {} {} \n' .format(round(date[4]),
round(date[5]), round(date[6]), sep=":"))
    fileID.write('Elevation:  {} \n' .format(round(date[1]), sep=":"))
    fileID.write('Azimuth:  {} \n' .format(round(date[2]), sep=":"))
    fileID.write('Azimuth Distance (km):  {} \n'
.format(round(date[3]), sep = ":"))

    if i == 3:
        fileID.write('Sun elevation:  {} \n' .format(data[4], sep=":"))

    else:
        fileID.write('Sun elevation:  {} \n' .format(data[4], sep=":"))
        fileID.write('\n')
    fileID.close()
```

1.1.2 Elevation_and_Azimuth

Elevation_and_Azimuth function calculates the elevation angle and azimuth of the satellite in the topocentric horizon coordinate system located at the observation site. The code is given below:

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot
th = 1
ph = 1
def Elevation_and_Azimuth(r, R, lstOS, Lat):

    #Elevation_and_Azimuth fonkisyonu, uydunun yükselme açısı ve yön
açısını gözlem noktasına yerleştirilmiş
    #yüzey-merkezli ufuk gözlem çerçevesine göre hesaplar.

    #a     - Uydunun yükseleme açısı, radyan
    #A     - Uydunun yön açısı, radyan
    #r     - Yer-merkezli ekvatoral koordinat sistemindeki uydunun
konum vektörü
    #R     - Yer-merkezli ekvatoral koordinat sistemindeki gözlem
bölgesinin konum vektörü
    #lstOS - Gözlem noktasının yıldız zamanı, derece
    #Lat   - Gözlem noktasının enlemi, derece

    #Yer-merkezli ekvatoral koordinat sisteminde gözlem noktasına göre
uydunun bağıl konum vektörü:
```

```
    rho = r - R

    #Yer-merkezli ekvatoral koordinat sisteminden yüzey-merkezli ufuk
gözlem çerçevesine değiştirme matrisi:
    th = (lstOS*180/math.pi)
    ph = (Lat*180/math.pi)

    Q_Xx = numpy.array([[-math.sin(th),           math.cos(th),
0],
                    [-math.sin(ph)*math.cos(th),  -
math.sin(ph)*math.sin(th),  math.cos(ph)],
                    [math.cos(ph)*math.cos(th),
math.cos(ph)*math.sin(th),  math.sin(ph)]])

    #Relativ pozisyon vektörünün yüzey-merkezli ufuk gözlem çerçevesine
göre yazılışı:
    rhoTH = numpy.linalg.inv(Q_Xx*numpy.linalg.inv(rho))

    #Relativ pozisyon vektörü yönündeki birim vektör:
    rhoTHdir = rhoTH/numpy.linalg.norm(rho)

    #Uydunun, gözlem noktasına yerleştirilmiş yüzey-merkezli ufuk
gözlem çerçevesinde
    #yükseliş açısının bulunuşu:
    a = math.asin(rhoTHdir[3])

    #Uydunun, gözlem noktasına yerleştirilmiş yüzey-merkezli ufuk
gözlem çerçevesinde
    #yerberi açısının bulunuşu:
    A = math.acos(rhoTHdir[2]/math.cos(a))
    if sign(sin(A)) == sign(rhoTHdir[1]/math.cos(a)):
            A=A
    else:
            A = 2*math.pi - A
Elevation_and_Azimuth(r, R, lstOS, Lat)
```

1.1.3 Analytic_Two_Body_Propagator

Analytic_Two_Body_Propagator function calculates the true anomaly after Dt amount of time in seconds. Other orbital elements defining the orientation of the orbital plane and the oritentation of the elliptical orbit on the orbital plane remain the same in Dt time interval. The code is given below:

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def Analytic_Two_Body_Propagator(kye0, Me0, n, Dt):

#Analytic_Two_Body_Propagator fonksiyonu, geçen Dt (saniye) zamanından
sonra doğru ayrıklığı hesaplar.
#Diğer yörünge elemanları, yörünge düzleminin dönmesini ve eliptik
yörüngenin düzlem üzerindeki dönüşünün geçen Dt zaman
#aralağında aynı kalışını tanımlar.
```

```
    #kye      - Klasik yörünge elemanları [h, i, Omega, omega, e, theta]
    #kye0     - Klasik yörünge elemanlarının başlangıç verileri
    #Me0      - Ortalama ayrıklık, birim radyan
    #n        - Ortalama hareket, birim radyan/saniye
    #Dt       - Yayılım için zaman aralığı, birim saniye
    #e        - Basıklık
    #theta    - Doğru ayrıklık, birim radyan

    #Yukarıdaki tüm açısal birimler radyan cinsindendir.

    e = 0
    Me0 = 5.5383
    n = 0.0011
    Dt = 3*24*3600

    Me = numpy.mod(Me0 + n*Dt,2*math.pi)
    theta = Kepler_Eqn_Solver_for_theta(Me,e)
    kye = kye0
    kye[6] = theta
Analytic_Two_Body_Propagator(kye0, Me0, n, Dt)
```

1.1.4 Kepler_Eqn_Solver_for_theta

Kepler_Eqn_Solver_for_theta function solves the Kepler equation to find true anomaly theta for given mean anomaly Me and eccentricity e.

```
import numpy
import scipy
import math
import sympy
from scipy import signal
from matplotlib import pyplot
from scipy.optimize import fsolve

def Kepler_Eqn_Solver_for_theta(Me, e):

    #Kepler_Eqn_Solver_for_theta fonksiyonu doğru ayrıklık değeri
theta'yı bulmak için verilen ortalama ayrıklık
    #değeri Me ve basıklık değeri e'yi kullanarak Kepler denklemini
çözer.

    #theta    -Doğru ayrıklık değeri, radyan
    #Me       -Ortalama ayrıklık değeri, radyan
    #e        -Basıklık, radyan

    #Kepler denklemi, f(E) = 0
    def f(E):
        KeplerEqn = E - e*math.sin(E) - Me
        return KeplerEqn
    #Kepler denkleminin çözümü, başlangıç tahmini Pi.
        E = fsolve(f,[math.pi])

    if math.tan(E/2.0)<0:
```

```
        theta = 2.0*math.atan(math.sqrt((1+e)/(1-e))*math.tan(E/2.0)) +
2.0*math.pi
    else:
        theta = 2.0*math.atan(math.sqrt((1+e)/(1-e))*math.tan(E/2.0))

Kepler_Eqn_Solver_for_theta(Me, e)
```

1.1.5 Visibility

Visibility function determines whether the satellite is visible from the obseravation site and provides the data for the visible pass.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def Visibility(r, R, lstOS, Lat, r_sun):
    #Visibility fonksiyonu uydunun gözlem noktasından görünür olup
olmadığını belirler ve
    #visible pass (görünür geçiş) için data (veri) sağlar.

    #v        -Eğer uydu gözlem noktasının üzerinden geçiyorsa 1,
geçmiyorsa 0.
    #r        -Yermerkezli ekvatoral çerçevedeki uydunun pozisyon
vektörü
    #R        -Yermerkezli ekvatoral çerçevedeki gözlem noktasının
pozisyon vektörü
    #lstOS    -Gözlem noktasının yıldız zamanı (derece)
    #Lat      -Gözlem noktasının enlemi (derece)
    #r_sun    -Güneş'in yermerkezli ekvatoral çerçevedeki pozisyon
vektörü
    #aSun     -Güneş'in yükselme açısı (radyan)
    #Asun     -Güneş'in yerberi açısı (radyan)
    #a        -Uydunun yükselme açısı (radyan)
    #A        -Uydunun yerberi açısı (radyan)
    #distance -Uydu ile gözlem noktasının arasındaki mesafe

    #Güneş'in yükselişini hesaplanması:
    [aSun, Asun] = Elevation_and_Azimuth(r_sun, R, lstOS, Lat)
    nu = los(r, r_sun)

    if  nu == 1 and aSun*180/math.pi < -6:
        v = 1
    else:
        v = 0

     #The visible pass data:
    [a, A] = Elevation_and_Azimuth(r, R, lstOS, Lat);
    distance = numpy.linalg.norm(r-R);
    data = [(a*180/math.pi), (A*180/math.pi), distance,
(aSun*180/math.pi)]
Visibility(r, R, lstOS, Lat, r_sun)
```

1.1.6 sv_from_coe

This function computes the state vector (r,v) from the classical orbital elements (coe).

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def sv_from_coe(coe,mu):
    #Bu fonksiyon durum vektörü (r,v) klasik yörünge elemanlarından
(kye) hesaplar.

    #mu   -kütleçekimsel parametre (km^3;s^2)
    #kye  -klasik yörünge elemanları:
        #h    -açısal momentum (km^2/s)
        #e    -basıklık
        #RA   -yükselme düğümü doğrultusu (rad)
        #incl -yörünge eğikliği (rad)
        #w    -argument of perigee (rad)
        #TA   -doğru açıklık (rad)
    #R3_w - Z-ekseni etrafında dönme matrisi w'ye doğru
    #R1_i - X-ekseni etrafında dönme matrisi i'ye doğru
    #R3_W - Z-ekseni etrafına dönme matrisi RA'ya doğru
    #Q_pX - Perifokaldan yermerkezli ekvatoral çerçeveye doğru olan
transfer matrisi
    #rp   - Perifokal çerçevedeki pozisyon vektörü (km)
    #vp   - Perifokal çerçevedeki hız vektörü (km/s)
    #r    - Yermerkezli ekvatoral çerçevedeki pozisyon vektörü (km)
    #v    - Yermerkezli ekvatoral çerçevedeki hız vektörü (km/s)


    h    = kye[1]
    incl = kye[2]
    RA   = kye[3]
    e    = kye[4]
    w    = kye[5]
    TA   = kye[6]

    #Denklemler 4.45 ve 4.46 (rp ve vp sütun vektörleridir.)
    sütunvektörü1 = numpy.array([[1],
                         [0],
                         [0]])
    sütunvektörü2 = numpy.array([0,
                         1,
                         0])


    rp = (h**2/mu) * (1/(1 + e*math.cos(TA))) *
(math.cos(TA)*sütunvektörü1 + math.sin(TA)*sütunvektörü2)
    vp = (mu/h) * (-math.sin(TA)*sütunvektörü1 + (e +
math.cos(TA))*sütunvektörü2)

    #Denklem 4.34:
    R1_i= numpy.array([ [1,  0,  0],
                    [0,  math.cos(incl),  math.sin(incl)],
                    [0, -math.sin(incl), math.cos(incl)]])
```

```
    #Denklem 4.34:
    R3_w = numpy.array([[math.cos(w),  math.sin(w),  0],
        [-math.sin(w),  math.cos(w),  0],
         [0,       0,    1]]);
    Q_pX = numpy.linalg.inv(R3_w*R1_i*R3_W)

    #Denklem 4.51 (r ve v sütun vektörleri):
    r = Q_pX*rp
    v = Q_pX*vp

    #r ve v'nin satır vektörlerine dönüştürülmesi:
    r = numpy.linalg.pinv(r)
    v = numpy.linalg.pinv(v)
sv_from_coe(coe,mu)
```

1.1.7 Solar_Position

Solar_Position function calculates the geocentric equatorial position vector of Sun at the given instant of time. This function is a modified form of the function solar_position.m of Curtis.

Solar position is important for two reasons. First, to able the visible passes of a satellite observer should located below the horizon line of the Sun is required. Second, the satellite reflects the Sun light and it becomes visible like that. So Sun has to be in the line of sight of satellite.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot


def Solar_Position(date):
    #Solar_Position foksiyonu, Güneş'in yermerkezli ekvatoral pozisyon
vektörünü
    #verilen anlık zamanda hesaplar.
    #Bu fonksiyon, Curtis'in solar_position.m fonksiyonun modifiye
edilmiş halidir.

    #date             -Anlık tarih, tarih vektörü olarak
    #UT               -Anlık evrensel zaman

    #Astronomik birim (km):
    AU = 149597870.691

    #Anlık Jülyen günü sayısı:
    UT = date[4] + date[5]/60 + date[6]/3600
    jd = J0(date[1],date[2],date[3]) + UT/24

    #J2000'den beri Jülyen günleri:
    n = jd - 2451545

    #J2000'den beri Jülyen yüzyılları:
```

```
    cy = n/36525

    #Ortalama ayrıklık (deg{:
    #Aşağıdaki satır 3rd edition Curtis'e göre güncelleştirilmiştir.
    M = 357.529 + 0.98560023*n
    M = numpy.mod(M,360)

    #Mean longitude (deg):
    #Aşağıdaki satır 3rd edition Curtis'e göre güncelleştirilmiştir.
    L = 280.459 + 0.98564736*n
    L = numpy.mod(L,360)

    #Açık ekliptik boylamı (deg):
    Lambda = L + 1.915*sind(M) + 0.020*sind(2*M)
    Lambda = numpy.mod(Lambda,360)

    #Ekliptiğin eğikliği (deg):
    #Aşağıdaki satır 3rd edition Curtis'e göre güncelleştirilmiştir.
    eps = 23.439 - 0.000000356*n

    #Dünya'dan Güneş'e doğru olan birim vektör:
    u = [math.cos(Lambda*math.pi/180),
math.sin(Lambda*math.pi/180)*math.cos(eps*math.pi/180),
math.sin(Lambda*math.pi/180)*math.sin(eps*math.pi/180)]

    #Dünya Güneş arası uzaklık (km):
    rS = (1.00014 - 0.01671*math.cos(M*math.pi/180) -
0.000140*math.cos(2*M*math.pi/180))*AU

    #Yermekezli pozisyon vektörü (km):
    r_S = rS*u
Solar_Position(date)
```

1.1.8 Position_of_Observation_Site

Position_of_Observation_Site function calculates the position vector of the observation site in geocentric equatorial frame.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def Position_of_Observation_Site(date,Observation_Site):

    #Position_of_Observation_Site fonksiyonu yermekezli ekvatoral
çerçevede bulunan
    #gözlem noktasının pozisyon vektörünü hesaplar

    #R               - Yermekezli ekvatoral çerçevede bulunan gözlem
noktasının pozisyonu
    #Observation_Site - Doğu boylamını içerir, gözlem noktasının enlemi
ve yüksekliği
    #date            - Anlık tarih
    #UT              - Anlık evrensel zaman (universal time)
    #EL              - Gözlem noktasının doğu boylamı, (derece)
```

```
    #Lat              - Gözlem noktasının enlemi, (derece)
    #lstOS            - Gözlem noktasının yıldız zamanı, (derece)
    #H                - Gözlem noktasının yüksekliği, (km)


    #Dünya'nın yarıçapı RE (km) ve Dünya'nın basıklığı (birimsiz)
    RE = 6378
    f = 0.00335

    #Gözlem noktası:
    EL = Observation_Site(1)
    Lat = Observation_Site(2)
    H = Observation_Site(3)

    #Anlık evrensel zaman (universal time):
    UT = date(4) + date(5)/60 + date(6)/3600

    #Gözlem noktası için yerel yıldız zamanı (Yermerkezli ekvatoral
çerçevede
    #gözlem noktasının açısal pozisyonunun bulunması).
    #Yerel yıldız zamanı LST çıktısının biriminin derece olduğuna
dikkat edilmesi gerekir.
    lstOS = LST(date(1),date(2),date(3),UT,EL)

    #Yermerkezli ekvatoral çerçevede gözlem noktasının pozisyonunun
    #(5.56)  Curtis, 3rd Ed. kullanılarak bulunması:
    th = deg2rad(lstOS)
    ph = deg2rad(Lat)

    Rx = (RE/sqrt(1-(2*f-f^2)*sin(ph)^2) + H)*cos(ph)*cos(th)
    Ry = (RE/sqrt(1-(2*f-f^2)*sin(ph)^2) + H)*cos(ph)*sin(th)
    Rz = (RE*(1-f)^2/sqrt(1-(2*f-f^2)*sin(ph)^2) + H)*sin(ph)

    R = [Rx, Ry, Rz]
Position_of_Observation_Site(date,Observation_Site)
```

1.1.9 Pass

Pass function checks whether the satellite passing over the observation site when the satellite at the position r and the observation site at the position R.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def Pass(r, R, lstOS, Lat):

    #Pass fonksiyonu, uydunun konumunun r olduğu ve gözlem noktasının R
olduğu zamanda
    #gözlem noktasının üzerinden geçip geçmediğini kontrol eder.

    #p      -Eğer uydu gözlem noktasının üzerinden geçiyorsa 1,
geçmiyorsa 0
    #r      -Yermerkezli ekvatoral çerçevedeki uydunun pozisyon vektörü
```

```
    #R      -Yermerkezli ekvatoral çerçevedeki gözlem noktasının
pozisyon vektörü
    #lstOS  -Gözlem noktasının yıldız zamanı (derece)
    #Lat    -Gözlem noktasının enlemi (derece)
    #a      -Uydunun yükselme açısı (radyan)
    #A      -Uydunun yön açısı (radyan)

    #Gözlem noktasına yerleştirilmiş olan
    #yüzey-merkezli ufuk koordinat sisteminedeki yükselme açısı ve yön
açısı
    [a, A] = Elevation_and_Azimuth(r, R, lstOS, Lat)

    #Uydunun gözlem noktasının üzerinden geçip geçmediğinin kontrol
edilişi,
    #a > 10 derece:
    if rad2deg(a) > 10:
        p = 1
    else:
        p = 0
Pass(r, R, lstOS, Lat)
```

1.1.10 LST

This function calculates the local sidereal time.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def LST(y, m, d, ut, EL):

    #Bu fonksiyon yıldız zamanını hesaplar.

    #lst  -yıldız zamanı (derece)
    #y    -yıl
    #m    -ay
    #d    -day
    #ut   -Evrensel Zaman
    #EL   -Doğu boylamı
    #j0   -Jülyen günü sayısı, 0'ıncı saat UT
    #j    -J2000'den beri olan yüzyıl sayısı
    #g0   -Greenwich yıldız zamanı (derece), 0'ıncı saat UT
    #gst  -Greenwich yıldız zamanı (derece), belirlenmiş UT

    #Denklem 5.48:
    j0 = J0(y, m, d)

    #Denklem 5.49:
    j = (j0 - 2451545)/36525

    #Denklem 5.50:
    g0 = 100.4606184 + 36000.77004*j + 0.000387933*j**2 - 2.583e-8*j**3

    #g0'nun 0 ile 360 arasında olması için düşürülmesi
    g0 = zeroTo360(g0);
```

```
    #Denklem 5.51:
    gst = g0 + 360.98564724*ut/24;

    #Denklem 5.52:
    lst = gst + EL;

    #lst'nin 0 ile 360 derece arasına düşürülmesi:
    lst = lst - 360*fix(lst/360);

    return

def zeroTo360(x):

    #Bu alt fonksiyon bir açıyı 0 ile 360 dereceye düşürür.

    #x - Düşürülecek açı (derece)
    #y - Düşürülen açı

    if (x >= 360):
        x = x - fix(x/360)*360;
    elif (x < 0):
        x = x - (fix(x/360) - 1)*360
LST(y, m, d, ut, EL)
```

1.1.11 los

This function uses the ECI position vectors of the satellite (r_sat) and the sun (r_sun) to determine whether the earth is in the line of sight between the two.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def los(r_sat, r_sun):

    #Bu fonksiyon uydunun ve güneşin ECI (Earth-Centered inital/Dünya-
Merkezli başlangıç)
    #pozisyon vektörlerini kullanarak Dünya'nın; uydunun ve Güneş'in
arasında, görüş alanında olup
    #olmadığını hesaplar.

    RE = 6378
    rsat = norm(r_sat)
    rsun = norm(r_sun)

    #Güneş'in ve uydunun pozisyon vektörlerinin arasındaki açı:
    theta = math.acosd(dot(r_sat, r_sun)/rsat/rsun)

    #Angle between the satellite position vector and the radial to the
point
    #of tangency with the earth of a line from the satellite:
    theta_sat = math.acosd(RE/rsat)

    #Angle between the sun position vector and the radial to the point
```

```
    #of tangency with the earth of a line from the sun:
    theta_sun = math.acosd(RE/rsun)

    #Güneş'ten uyduya doğru olan bir çizginin Dünya ile keşistiğinin
bulunması:

    if theta_sat + theta_sun <= theta:
        light_switch = 0;   #evet
    else:
        light_switch = 1;   #hayır
los(r_sat, r_sun)
```

1.1.12 J0

This function computes the Julian day number at 0 UT for any year between 1900 and 2100 using Equation 5.48.

The Julian day number is the number of days since noon UT on January 1, 4713 BC.

The origin of this time scale is placed in antiquity so that, except for prehistoric events, we do not have to deal with positive and negative dates. The Julian day count is uniform and continuous and does not involve leap years or different numbers of days in different months. The number of days between two events is found by simply subtracting the Julian day of one from that of the other. The JD begins at noon rather than at midnight so that astronomers observing the heavens at night would not have to deal with a change of date during their watch.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

def J0(year, month, day):

    #Bu fonksiyon Jülyen günü sayısını 0 UT ve 1900 ile 2100 arasındaki
herhangi bir yıl için
    #Denklem 5.48'i kullanarak hesaplar.

    #j0     -Jülyen günü, 0 UT (Evrensel Zaman/Universal Time)
    #year   -aralık: 1901 - 2099
    #month  -aralık: 1 - 12
    #day    -aralık: 1 - 31

j0 = 367*year - round(7*(year + round((month + 9)/12))/4) +
round(275*month/9) + day + 1721013.5;
print (j0)
J0(year, month, day)
```

1.1.13 Average_J2_Perturbation

Average_J2_Perturbation function calculates the average rate of change of Omega and omega due to oblateness of Earth.

Due to Earth rotation, there is a centripetal acceleration in every particle that is a spherically symmetric mass distribution. This particles rotates with the same angular velocity. But their distances is different. So a particle which far away from rotation axis has more angular velocity than closer ones. Hence, according to Newton's second law to create this kind of centripetal acceleration there has to be more centripetal force in far particles. In that situtaion, in the existance of fluid construction, every time there will be collapse in equator. For this reason Earth shape will not be perfect and bulging at the equator and flattened at the poles. Zonal harmonics are used to express the cylindrical symmetrical contraction area of the world.

Zonal harmonics are unitless and not universal; that is, each planet has its own unique

Harmonic coefficients. In this context, harmonic coefficients of any mathematical inference not the result of observing orbital movements around the planets.

The harmonic values of the towel are clear that the harmonic values of J2 are clearly the most dominant values and that the effect of other coefficients can be neglected in situation. Therefore, in the calculation of the trajectory perturbation of the Earth only J2 perturbation is used. With the J2 perturbation right ascension node and argument of perigee changed effectively in the calculation of orbit analysis precision.

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot

    #AvarajJ2Pertürbasyonu Omega ve omega'nın Dünya'nın basıklığına
göre ortalama değişimini hesaplar.

    #kye_dot    - Klasik yörünge elemanlarının zamana göre türevi
    #kyeTwoBody - Two-body problemden elde edilen klasik yörünge
elemanları
    #kye        - Klasik Yörünge Elemanları
    #Dt         - TLE zamanından hesaplanan zaman
    #mu         - Kütleçekim Parametresi, mu, birimi;  km^3/s^2
    #J2         - İkinci kuşak harmonikleri (birimsiz), R yarıçap (km)


J2 = 0.00108263
RE = 6378


def AvarajJ2Pertürbasyonu_(kyeTwoBody, dt, mu):
```

```
    h = kyeTwoBody[0]
    i = kyeTwoBody[1]
    e = kyeTwoBody[3]

    #Yarı-büyük eksenin hesaplanışı
    a = h**2/mu*1/(1-e**2)


    AvarajJ2Pertürbasyonu_(kyeTwoBody, Dt, mu)
    h = kyeTwoBody[0]
    i = kyeTwoBody[1]
    e = kyeTwoBody[3]
    a = h**2/mu*1/(1-e**2)

#Yükseklme açısı, yön açısı ve yerberi açısının ortalama değişim
değerleri Curtis'in 3rd Ed (4.52) ve (4.53) verilmiştir.
    Omega_dot = - (3/2*(math.sqrt(398600)*J2*RE**2)/((1-
e**2)**2*a**(7/2)))**math.cos(i)
    omega_dot = - (3/2*(math.sqrt(398600)*J2*RE**2)/((1-
e**2)**2*a**(7/2)))**(5/2*math.sin(i)**2-2)

    kye_dot = [0,0,Omega_dot,0,omega_dot]
    new_kye_dot = [i * Dt for i in kye_dot]

#Klasik yörünge elemanlarının J2 pertürbasyonun etkisi altında
hesaplanışı
    print(kye)
    print(kyeTwoBody[5])
    kye5 = kyeTwoBody[5]
AvarajJ2Pertürbasyonu_(kyeTwoBody, dt, mu)
```

1.1.14 Visible_Passes

This is the main code and unfinished code in Python. But in MATLAB whole codes are finished and worked correctly by the following persons next to this writing. Visible_Passes function is the main function finding the visible passes. This code is written by Tahsin Çağrı Şişman and closely related with the code developed for the paper "Uyduların Belirli Bir Bölgeden Görünür Geçişlerinin Saptanması" by Mehmet Fatih Ertürk, Şahin Ulaş Köprücü, Uzay Tuğcular, İbrahim Arda, Yılmaz Barış Erkan, and Tahsin Çağrı Şişman [3]. In writing this code, for some parts M-files provided by Howard Curtis are directly used (sv_from_coe.m, LST.m, los.m) or modified (solar_position.m). These uses are cited at the related points.

```
import numpy
from TLE_to_COE_and_t0 import TLE_to_COE_and_t0
from Position_of_Observation_Site import Position_of_Observation_Site
import scipy
import math
from scipy import signal
from matplotlib import pyplot
from datetime import date
```

```
from datetime import datetime
from datetime import timedelta

#def Visible_Passes():

    #Visibe_Passes kodu, görünür geçişleri bulan ana koddur. Bu kod,
    #başta Tahsin Çağrı Şişman tarafından yazılmıştır ve
    #"Uyduların Belirli Bir Bölgeden Görünür Geçişlerinin Saptanması"
    #bildirisi için geliştirilen kod ile ilişkili olarak ilgili olan
kişiler
    #Mehmet Fatih Ertürk, Şahin Ulaş Köprücü, Uzay Tuğcular, İbrahim
Arda,
    #Yılmaz Barış Erkan, and Tahsin Çağrı Şişman tarafından
yazılmıştır.

    #Bu kod yazılırken, bazı kısımlarda Howard Curtis tarafından
sağlanan
    #Matlab dosyaları direk kullanılmıştır, (sv_from_coe.m, LST.m,
los.m) yada
    #modifiye edilerek kullanılmıştır (solar_position.m).
    #Bu kullanımlar ilgili noktalarda gösterilmektedir.

    #kye0             -TLE verilerinden alınan klasik yörünge
elemanları
    #                  [h,i,Omega,e,omega,theta]
    #year             -TLE çağının yılı
    #day0             -TLE çağı için gün, kesri ile
    #Me0              -TLE çağındaki ortalama açıklık
    #n                -TLE çağındaki ortalama hareket
    #Observation_Site -Doğu boylamını içerir, gözlem noktasının enlemi
ve
    #                  yüksekliği
    #EL               -Gözlem noktasının doğu boylamı, derece
    #Lat              -Gözlem noktasının enlemi, derece
    #H                -Gözlem noktasının yükseliği, km
    #Delta_t          -Görünür geçişlerin incelendiği süreç
    #r                -Uydunun yermerkezli ekvatoral çerçevedeki
poziyon
    #                  vektörü
    #v                -Uydunun yermerkezli ekvatoral çerçevedeki hız
vektörü
    #R                -Gözlem noktasının yer-merkezli ekvatoral
çerçevedeki
    #                  pozisyon vektörü
    #lstOS            -Gözlem noktasının yıldız zamanı, derece
    #p                -Eğer uydu gözlem noktasının üzerinden geçiyorsa
1,
    #                  geçmiyorsa 0
    #r_sun            -Güneş'in yermerkezli ekvatoral çerçevedeki
pozisyon
    #                  vektörü
    #visible          -Eğer uydu gözlem noktasının üzerinde görünür
geçişe
    #                  sahipse 1, değilse 0
    #a                -Uydunun yükselme açısı, radyan
    #data             -Görünür geçiş verisi; date, time, elevation,
    #                  azimuth, distance, elevation of the Sun
```

```
    #TLE_to_COE_and_t0 fonksiyonunu kullanarak TLE datalarının okunması
ve
    #klasik yörünge elemanlarına çevrilişi.
    #Açısal çıktılar radyan cinsindendir.
#kye = TLE_to_COE_and_t0.TLE_to_COE_and_to()[0]
year = TLE_to_COE_and_t0.TLE_to_COE_and_t0()[6]
day = TLE_to_COE_and_t0.TLE_to_COE_and_t0()[2]
Me = TLE_to_COE_and_t0.TLE_to_COE_and_t0()[3]
n = TLE_to_COE_and_t0.TLE_to_COE_and_t0()[4]
epoch = TLE_to_COE_and_t0.TLE_to_COE_and_t0()[5]
#print (kye)


    #Kütleçekimsel parametre, km^3/s^2
mu = 398600

    #Gözlem noktası; Ankara, THKÜ kampüs alanı enlem 39.9455 derece
kuzey,
    #32.6890 derece doğu boylamı ve yükseklik 0.810 km.
EL = 32.6890
Lat = 39.9455
H = 0.810
def Position_of_Observation_Site(date,Observation_Site):
        Observation_Site = numpy.array([EL, Lat, H])

    #Output(çıktı) text dosyasının açılması
fileID = open('Visible_pass_output.txt','w+')
    #TLE tarihinin ve zamanının Output(çıktı) dosyasına yazılması


a = date.toordinal(date(year,1,day))+366

    #TLE'den sonra 3 günlük görünür geçişlerin hesaplanışı
Delta_t = 3*24*3600
visibleOld = 0

for Dt in range(0, Delta_t):
    #Yörüngenin yörünge düzleminde yayılması
    def Analytic_Two_Body_Propagator(kye0, Me0, n, Dt):
        kye = numpy.array([kye0, Me0, n, Dt])
    #J2 pertürbasyonun etkisi altında klasik yörünge elemanlarının
hesaplanışı
        def AvarajJ2Pertürbasyonu_(kyeTwoBody, dt, mu):
            kye = numpy.array[(kyeTwoBody, dt, mu)]
    #Yermerkezli ekvatoral çerçevede uydunun durum vektörlerinin
hesaplanışı.
    #Algorithma 4.5 açısal elemanların radyan cinsinden olmasını
söyler.
    #sv_from_coe.m dosyası Curtis'ten alınmıştır.
            def sv_from_coe(coe,mu):
                [r, v] = numpy.array([coe, mu])

    #
```

2. Second Way of Visible Passes Code

```
import numpy
import scipy
import math
from scipy import signal
from matplotlib import pyplot
from datetime import datetime,date
from datetime import timedelta
import time


def TLE_to_COE_and_t0():

    #TLE_to_COE_and_t0 fonksiyonu TLE verisini içeren TLE dosyasını
okur ve
    #bu TLE verilerini klasik yörünge elemanlarına çevirir, başlangıç
zamanı da kapsar.

    #Fonksiyonun çıktıları klasik yörünge elemanları (özgül açısal
momentum,
    #eğiklik, yükselme düğümü açısı, basıklık, perigee argümanı, doğru
açıklık),
    #TLE çağının yılı, TLE çağı için gün, gün fraksiyonu ile, ortalama
açıklık ve
    #ortalama hareket

    #Output için, tüm açısal birimler radyan cinsindendir.

    #kye0             -TLE datanın klasik yörüge elemanları
[h,i,Omega,e,omega,theta]
    #year             -TLE çağının yılı
    #day              -TLE çağı için gün, fraksiyonu ile birlikte
    #Me               -TLE çağı için Ortalama açıklık
    #n                -TLE çağı için Ortalama hareket
    #Observation_Site -Doğu boylamı içeren, gözlem noktasının enlemi ve
yüksekliği
    #date             -Anlık tarih
    #UT               -Anlık evrensel zaman (Universal time)
    #EL               -Gözlem noktasının Doğu boylamı (derece)
    #Lat              -Gözlem noktasının enlemi (derece)
    #lstOS            -Gözlem noktasının yıldız zamanı (derece)
    #H                -Gözlem noktasının yüksekliği (km)

    #TLE dosyası tle.txt okunması. Uygun bir tle.text dosyası
oluşturmak için,
    #space-track.org dan TLE verisi elde edin ve kopyalayın ve tle.text
    #dosyasına yapıştırın.
    #Aşağıdaki bölümde, TLE verisinin ilk satırı Line 1 cell array
olarak okunur ve
    #TLE verisinin ikinci satırı Line 2 cell array olarak okunur.

    fileID = open('tles.txt', 'r')
    fileID.seek(18)
    epoch = float(fileID.read(5))
    print (epoch)
    fileID.seek(40)
    n0dot = 2.0*(float(fileID.read(3))/1000000)
```

```
    fileID.seek(45)
    n0doubledot1 = 6.0*(float(fileID.read(1)))
    fileID.seek(51)
    n0doubledot2 = 10**float(fileID.read(1))
    n0doubledot = n0doubledot1*n0doubledot2
    fileID.seek(54)
    Bstar1 = float(fileID.read(5))
    fileID.seek(60)
    Bstar2 = 10**(-float(fileID.read(1)))
    Bstar = Bstar1*Bstar2
    fileID.seek(80)
    i = float(fileID.read(7))*math.pi/180.0
    fileID.seek(88)
    Omega = (float(fileID.read(8)))*math.pi/180.0
    fileID.seek(100)
    e = (float(fileID.read(4)))/(10**7)
    fileID.seek(105)
    omega = (float(fileID.read(8)))*math.pi/180.0
    fileID.seek(114)
    Me = (float(fileID.read(8)))*math.pi/180.0
    fileID.seek(123)
    # Ortalama hareket, rad/s biriminde
    n = (float(fileID.read(7)))*2.0*math.pi/(24.0*3600.0)

    #Klasik yörünge elemanlarının bulunuşu, bunlar; h, i, Omega, e,
omega ,theta
    #TLE data çıktıları. i, Omega, e, omega TLE dosyasından okunabilir.
h ve theta,
    #n ve Me'ye göre bulunur.

    #Kütleçekimsel parametre mu, km^3/s^2:
    mu = 398600

    #h'ın n'den elde edilişi:
    h = (mu**2/n)**(1/3)*math.sqrt(1-e**2)


    #Kepler denklemini çözerek theta'nın Me'den elde edilişi:

    def f(E):
            KeplerEqn = E - e*math.sin(E) - Me
            return KeplerEqn
    #Kepler denkleminin çözümü, başlangıç tahmini Pi.
            E = fsolve(f,[math.pi])

            if math.tan(E/2.0)<0:
                theta = 2.0*math.atan(math.sqrt((1+e)/(1-
e))*math.tan(E/2.0)) + 2.0*math.pi;
            else:
                theta = 2.0*math.atan(math.sqrt((1+e)/(1-
e))*math.tan(E/2.0));
                #Klasik yörünge elemanları(kye). Ortalama hareketin
birimi,
                #rad/s ve bütün açılar radyan cinsinden.
                kye = numpy.array([h, i, Omega, e, omega, theta])
                print (kye)
```

```
    #Çağ, Evrensel Zamanı tanımlar (Greenwich'teki güneş zamanı).
Aşağıda,
    #çağ yıla çevrilmiştir, yılın günü gün kesri ile

    year = 2000 + (epoch - numpy.mod(epoch,1000))/1000

    day = numpy.mod(epoch,1000)

    #Bu fonksiyon yıldız zamanını hesaplar.

    #lst  -yıldız zamanı (derece)
    #y    -yıl
    #m    -ay
    #d    -day
    #ut   -Evrensel Zaman
    #EL   -Doğu boylamı
    #j0   -Jülyen günü sayısı, 0'ıncı saat UT
    #j    -J2000'den beri olan yüzyıl sayısı
    #g0   -Greenwich yıldız zamanı (derece), 0'ıncı saat UT
    #gst  -Greenwich yıldız zamanı (derece), belirlenmiş UT

    #Denklem 5.48:
    #Bu fonksiyon Jülyen günü sayısını 0 UT ve 1900 ile 2100 arasındaki
herhangi bir yıl için
    #Denklem 5.48'i kullanarak hesaplar.

    #j0     -Jülyen günü, 0 UT (Evrensel Zaman/Universal Time)
    #year   -aralık: 1901 - 2099
    #month  -aralık: 1 - 12
    #day    -aralık: 1 - 31
    month=1

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


    #Denklem 5.49:
    j = (j0 - 2451545)/36525

    #Denklem 5.50:
    g0 = 100.4606184 + 36000.77004*j + 0.000387933*j**2 - 2.583e-8*j**3

    #g0'nun 0 ile 360 arasında olması için düşürülmesi
    #g0 = zeroTo360(g0)
    global date
    datenumnew= date.toordinal(date(int(year),int(1),int(30)))+1/8+366
    daysnew = datenumnew % 1
    hoursnew = daysnew % 1 * 24
    minutesnew = hoursnew % 1 * 60
    secondsnew = minutesnew % 1 * 60
    UTnew = hoursnew + minutesnew/60 + secondsnew/3600
    datenew=datetime.fromordinal(int(datenumnew))+
timedelta(days=int(daysnew)) +timedelta(hours=int(hoursnew)) +
timedelta(minutes=int(minutesnew))+
timedelta(seconds=round(secondsnew)) - timedelta(days=366)
    #Denklem 5.51:
    gst = g0 + 360.98564724*UTnew/24;
    #Dünya'nın yarıçapı RE (km) ve Dünya'nın basıklığı (birimsiz)
```

```
    RE = 6378
    f = 0.00335

    #Gözlem noktası:
    EL = 32.6890
    Lat = 39.9455
    H = 0.810

    #Denklem 5.52:
    lst = gst + EL;

    #lst'nin 0 ile 360 derece arasına düşürülmesi:
    lst = lst - 360*numpy.fix(lst/360);




        #Position_of_Observation_Site fonksiyonu yermekezli ekvatoral
çerçevede bulunan
    #gözlem noktasının pozisyon vektörünü hesaplar

    #R                - Yermekezli ekvatoral çerçevede bulunan gözlem
noktasının pozisyonu
    #Observation_Site - Doğu boylamını içerir, gözlem noktasının enlemi
ve yüksekliği
    #date             - Anlık tarih
    #UT               - Anlık evrensel zaman (universal time)
    #EL               - Gözlem noktasının doğu boylamı, (derece)
    #Lat              - Gözlem noktasının enlemi, (derece)
    #lstOS            - Gözlem noktasının yıldız zamanı, (derece)
    #H                - Gözlem noktasının yüksekliği, (km)


    #Dünya'nın yarıçapı RE (km) ve Dünya'nın basıklığı (birimsiz)

    #Anlık evrensel zaman (universal time):

    #Gözlem noktası için yerel yıldız zamanı (Yermerkezli ekvatoral
çerçevede
    #gözlem noktasının açısal pozisyonunun bulunması).
    #Yerel yıldız zamanı LST çıktısının biriminin derece olduğuna
dikkat edilmesi gerekir.
#lstOS = LST(year,date(2),date(3),UT,EL)

    #Yermerkezli ekvatoral çerçevede gözlem noktasının pozisyonunun
    #(5.56)  Curtis, 3rd Ed. kullanılarak bulunması:

    th = int(lst*math.pi/180)
    ph = int(Lat*math.pi/180)
    f=int(f)

    Rx = (RE/math.sqrt(1-(2*f-f**2)*math.sin(ph)**2) +
H)*math.cos(ph)*math.cos(th)
    Ry = (RE/math.sqrt(1-(2*f-f**2)*math.sin(ph)**2) +
H)*math.cos(ph)*math.sin(th)
```

```
    Rz = (RE*(1-f)**2/math.sqrt(1-(2*f-f**2)*math.sin(ph)**2) +
H)*math.sin(ph)

    R = [Rx, Ry, Rz]

    fileID = open('Visible_pass_output.txt','w+')
    datenum= date.toordinal(date(int(year),int(12),int(27)))+1/8+366
    print (datenum)
    days = datenum % 1
    hours = days % 1 * 24
    minutes = hours % 1 * 60
    seconds = minutes % 1 * 60
    date=datetime.fromordinal(int(datenum))+ timedelta(days=int(days))
+timedelta(hours=int(hours)) + timedelta(minutes=int(minutes))+
timedelta(seconds=round(seconds)) - timedelta(days=366)

    fileID.write('Date and time of TLE: {} \n'.format(date))


    #Calculating the visible passes for three days time after TLE
    Delta_t = 3*24*3600
    visibleOld = 0

    for Dt in range(0,3*24*3600):


        #Analytic_Two_Body_Propagator fonksiyonu, geçen Dt (saniye)
zamanından sonra doğru ayrıklığı hesaplar.
        #Diğer yörünge elemanları, yörünge düzleminin dönmesini ve
eliptik yörüngenin düzlem üzerindeki dönüşünün geçen Dt zaman
        #aralağında aynı kalışını tanımlar.

        #kye     - Klasik yörünge elemanları [h, i, Omega, omega, e,
theta]
        #kye0    - Klasik yörünge elemanlarının başlangıç verileri
        #Me0     - Ortalama ayrıklık, birim radyan
        #n       - Ortalama hareket, birim radyan/saniye
        #Dt      - Yayılım için zaman aralığı, birim saniye
        #e       - Basıklık
        #theta   - Doğru ayrıklık, birim radyan

        #Yukarıdaki tüm açısal birimler radyan cinsindendir.

        e = 0
        Me0 = 5.5383
        n = 0.0011
        Dt = 3*24*3600

        Me = numpy.mod(Me0 + n*Dt,2*math.pi)


            #Kepler_Eqn_Solver_for_theta fonksiyonu doğru ayrıklık
değeri theta'yı bulmak için verilen ortalama ayrıklık
            #değeri Me ve basıklık değeri e'yi kullanarak Kepler
denklemini çözer.

        #theta   -Doğru ayrıklık değeri, radyan
        #Me      -Ortalama ayrıklık değeri, radyan
```

```
        #e       -Basıklık, radyan

        #Kepler denklemi, f(E) = 0
        def f(E):
            KeplerEqn = E - e*math.sin(E) - Me
            return KeplerEqn
        #Kepler denkleminin çözümü, başlangıç tahmini Pi.
            E = fsolve(f,[math.pi])

            if math.tan(E/2.0)<0:
                theta = 2.0*math.atan(math.sqrt((1+e)/(1-
e))*math.tan(E/2.0)) + 2.0*math.pi
            else:
                theta = 2.0*math.atan(math.sqrt((1+e)/(1-
e))*math.tan(E/2.0))
                kye = numpy.array([h, i, Omega, e, omega, theta])
                print(kye)
                kye[5] = theta

            #AvarajJ2Pertürbasyonu Omega ve omega'nın Dünya'nın
basıklığına göre ortalama değişimini hesaplar.

    #kye_dot    - Klasik yörünge elemanlarının zamana göre türevi
    #kyeTwoBody - Two-body problemden elde edilen klasik yörünge
elemanları
    #kye        - Klasik Yörünge Elemanları
    #Dt         - TLE zamanından hesaplanan zaman
    #mu         - Kütleçekim Parametresi, mu, birimi;  km^3/s^2
    #J2         - İkinci kuşak harmonikleri (birimsiz), R yarıçap (km)


                J2 = 0.00108263
                RE = 6378
                h = kye[0]
                i = kye[1]
                e = kye[3]

        #Yarı-büyük eksenin hesaplanışı
                a = h**2/mu*1/(1-e**2)


                h = kye[0]
                i = kye[1]
                e = kye[3]
                a = h**2/mu*1/(1-e**2)

        #Yükseklme açısı, yön açısı ve yerberi açısının ortalama
değişim değerleri Curtis'in 3rd Ed (4.52) ve (4.53) verilmiştir.
                Omega_dot = - (3/2*(math.sqrt(398600)*J2*RE**2)/((1-
e**2)**2*a**(7/2)))**math.cos(i)
                omega_dot = - (3/2*(math.sqrt(398600)*J2*RE**2)/((1-
e**2)**2*a**(7/2)))**(5/2*math.sin(i)**2-2)

                kye_dot = [0,0,Omega_dot,0,omega_dot]
                new_kye_dot = [i * Dt for i in kye_dot]

        #Klasik yörünge elemanlarının J2 pertürbasyonun etkisi altında
hesaplanışı
```

```
                kye5 = kye[5]

        #Bu fonksiyon durum vektörü (r,v) klasik yörünge elemanlarından
(kye) hesaplar.

    #mu   -kütleçekimsel parametre (km^3;s^2)
    #kye  -klasik yörünge elemanları:
        #h    -açısal momentum (km^2/s)
        #e    -basıklık
        #RA   -yükselme düğümü doğrultusu (rad)
        #incl -yörünge eğikliği (rad)
        #w    -argument of perigee (rad)
        #TA   -doğru açıklık (rad)
    #R3_w - Z-ekseni etrafında dönme matrisi w'ye doğru
    #R1_i - X-ekseni etrafında dönme matrisi i'ye doğru
    #R3_W - Z-ekseni etrafına dönme matrisi RA'ya doğru
    #Q_pX - Perifokaldan yermerkezli ekvatoral çerçeveye doğru olan
transfer matrisi
    #rp   - Perifokal çerçevedeki pozisyon vektörü (km)
    #vp   - Perifokal çerçevedeki hız vektörü (km/s)
    #r    - Yermerkezli ekvatoral çerçevedeki pozisyon vektörü (km)
    #v    - Yermerkezli ekvatoral çerçevedeki hız vektörü (km/s)


                h = kye[1]
                incl = kye[2]
                RA   = kye[3]
                e    = kye[4]
                w    = kye[5]
                TA   = kye[6]

    #Denklemler 4.45 ve 4.46 (rp ve vp sütun vektörleridir.)
                sütunvektörü1 = numpy.array([[1],
                          [0],
                          [0]])
                sütunvektörü2 = numpy.array([0,
                          1,
                          0])


                rp = (h**2/mu) * (1/(1 + e*math.cos(TA))) *
(math.cos(TA)*sütunvektörü1 + math.sin(TA)*sütunvektörü2)
                vp = (mu/h) * (-math.sin(TA)*sütunvektörü1 + (e +
math.cos(TA))*sütunvektörü2)

    #Denklem 4.34:
                R1_i= numpy.array([ [1,  0,  0],
                    [0,  math.cos(incl),  math.sin(incl)],
                    [0, -math.sin(incl), math.cos(incl)]])

    #Denklem 4.34:
                R3_w = numpy.array([[math.cos(w),  math.sin(w),  0],
        [-math.sin(w),  math.cos(w),  0],
         [0,       0,    1]]);
                Q_pX = numpy.linalg.inv(R3_w*R1_i*R3_W)

    #Denklem 4.51 (r ve v sütun vektörleri):
                r = Q_pX*rp
```

```
                v = Q_pX*vp


    #r ve v'nin satır vektörlerine dönüştürülmesi:
                r = numpy.linalg.pinv(r)
                v = numpy.linalg.pinv(v)

                datenum=
date.toordinal(date(year,1,day))+366+Dt/(24*3600)
                print (datenum)
                days = datenum % 1
                hours = days % 1 * 24
                minutes = hours % 1 * 60
                seconds = minutes % 1 * 60
                date=datetime.fromordinal(int(datenum))+
timedelta(days=int(days)) +timedelta(hours=int(hours)) +
timedelta(minutes=int(minutes))+ timedelta(seconds=round(seconds)) -
timedelta(days=366)

                RE = 6378
                f = 0.00335

        #Gözlem noktası:
                EL = 32.6890
                Lat = 39.9455
                H = 0.810

        #Anlık evrensel zaman (universal time):
                UT = hours + minutes/60 + seconds/3600

    #Gözlem noktası için yerel yıldız zamanı (Yermerkezli ekvatoral
çerçevede
    #gözlem noktasının açısal pozisyonunun bulunması).
    #Yerel yıldız zamanı LST çıktısının biriminin derece olduğuna
dikkat edilmesi gerekir.

    #Yermerkezli ekvatoral çerçevede gözlem noktasının pozisyonunun
    #(5.56)  Curtis, 3rd Ed. kullanılarak bulunması:
                th = lst*math.pi/180
                ph = Lat*math.pi/180

                Rx = (RE/math.sqrt(1-(2*f-f**2)*math.sin(ph)**2) +
H)*math.cos(ph)*math.cos(th)
                Ry = (RE/math.sqrt(1-(2*f-f**2)*math.sin(ph)**2) +
H)*math.cos(ph)*math.sin(th)
                Rz = (RE*(1-f)**2/math.sqrt(1-(2*f-
f**2)*math.sin(ph)**2) + H)*math.sin(ph)

                R = [Rx, Ry, Rz]

         #Pass fonksiyonu, uydunun konumunun r olduğu ve gözlem
noktasının R olduğu zamanda
    #gözlem noktasının üzerinden geçip geçmediğini kontrol eder.

    #p      -Eğer uydu gözlem noktasının üzerinden geçiyorsa 1,
geçmiyorsa 0
    #r      -Yermerkezli ekvatoral çerçevedeki uydunun pozisyon vektörü
```

```
    #R      -Yermerkezli ekvatoral çerçevedeki gözlem noktasının
pozisyon vektörü
    #lstOS  -Gözlem noktasının yıldız zamanı (derece)
    #Lat    -Gözlem noktasının enlemi (derece)
    #a      -Uydunun yükselme açısı (radyan)
    #A      -Uydunun yön açısı (radyan)

    #Gözlem noktasına yerleştirilmiş olan
    #yüzey-merkezli ufuk koordinat sisteminedeki yükselme açısı ve yön
açısı
        #Elevation_and_Azimuth fonkisyonu, uydunun yükselme açısı ve
yön açısını gözlem noktasına yerleştirilmiş
    #yüzey-merkezli ufuk gözlem çerçevesine göre hesaplar.

    #a     - Uydunun yükseleme açısı, radyan
    #A     - Uydunun yön açısı, radyan
    #r     - Yer-merkezli ekvatoral koordinat sistemindeki uydunun
konum vektörü
    #R     - Yer-merkezli ekvatoral koordinat sistemindeki gözlem
bölgesinin konum vektörü
    #lstOS - Gözlem noktasının yıldız zamanı, derece
    #Lat   - Gözlem noktasının enlemi, derece

    #Yer-merkezli ekvatoral koordinat sisteminde gözlem noktasına göre
uydunun bağıl konum vektörü:
                rho = r - R

    #Yer-merkezli ekvatoral koordinat sisteminden yüzey-merkezli ufuk
gözlem çerçevesine değiştirme matrisi:


                Q_Xx = numpy.array([[-math.sin(th),
math.cos(th),          0],
                   [-math.sin(ph)*math.cos(th),  -
math.sin(ph)*math.sin(th),  math.cos(ph)],
                   [math.cos(ph)*math.cos(th),
math.cos(ph)*math.sin(th),  math.sin(ph)]])

    #Relativ pozisyon vektörünün yüzey-merkezli ufuk gözlem çerçevesine
göre yazılışı:
                rhoTH = numpy.linalg.inv(Q_Xx*numpy.linalg.inv(rho))

    #Relativ pozisyon vektörü yönündeki birim vektör:
                rhoTHdir = rhoTH/numpy.linalg.norm(rho)

    #Uydunun, gözlem noktasına yerleştirilmiş yüzey-merkezli ufuk
gözlem çerçevesinde
    #yükseliş açısının bulunuşu:
                a = math.asin(rhoTHdir[3])

    #Uydunun, gözlem noktasına yerleştirilmiş yüzey-merkezli ufuk
gözlem çerçevesinde
    #yerberi açısının bulunuşu:
                A = math.acos(rhoTHdir[2]/math.cos(a))
                if sign(sin(A)) == sign(rhoTHdir[1]/math.cos(a)):
                    A=A
                else:
                    A = 2*math.pi - A
```

```
    #Uydunun gözlem noktasının üzerinden geçip geçmediğinin kontrol
edilişi,
    #a > 10 derece:
                if a*180/math.pi > 10:
                    p = 1
                else:
                    p = 0

                if p==1:
            #Solar_Position foksiyonu, Güneş'in yermerkezli ekvatoral
pozisyon vektörünü
    #verilen anlık zamanda hesaplar.
    #Bu fonksiyon, Curtis'in solar_position.m fonksiyonun modifiye
edilmiş halidir.

    #date             -Anlık tarih, tarih vektörü olarak
    #UT               -Anlık evrensel zaman

    #Astronomik birim (km):
                    AU = 149597870.691

    #Anlık Jülyen günü sayısı:
                    UT = date[3] + date[4]/60 + date[5]/3600
                    jd = J0(date[0],date[1],date[2]) + UT/24

    #J2000'den beri Jülyen günleri:
                    n = jd - 2451545

    #J2000'den beri Jülyen yüzyılları:
                    cy = n/36525

    #Ortalama ayrıklık (deg{:
    #Aşağıdaki satır 3rd edition Curtis'e göre güncelleştirilmiştir.
                    M = 357.529 + 0.98560023*n
                    M = numpy.mod(M,360)

    #Mean longitude (deg):
    #Aşağıdaki satır 3rd edition Curtis'e göre güncelleştirilmiştir.
                    L = 280.459 + 0.98564736*n
                    L = numpy.mod(L,360)

    #Açık ekliptik boylamı (deg):
                    Lambda = L + 1.915*sind(M) + 0.020*sind(2*M)
                    Lambda = numpy.mod(Lambda,360)

    #Ekliptiğin eğikliği (deg):
    #Aşağıdaki satır 3rd edition Curtis'e göre güncelleştirilmiştir.
                    eps = 23.439 - 0.000000356*n

    #Dünya'dan Güneş'e doğru olan birim vektör:
                    u = [math.cos(Lambda*math.pi/180),
math.sin(Lambda*math.pi/180)*math.cos(eps*math.pi/180),
math.sin(Lambda*math.pi/180)*math.sin(eps*math.pi/180)]

    #Dünya Güneş arası uzaklık (km):
                    rS = (1.00014 - 0.01671*math.cos(M*math.pi/180) -
0.000140*math.cos(2*M*math.pi/180))*AU
```

```
    #Yermekezli pozisyon vektörü (km):
                    r_S = rS*u

            #Visibility fonksiyonu uydunun gözlem noktasından görünür
olup olmadığını belirler ve
    #visible pass (görünür geçiş) için data (veri) sağlar.

    #v        -Eğer uydu gözlem noktasının üzerinden geçiyorsa 1,
geçmiyorsa 0.
    #r        -Yermerkezli ekvatoral çerçevedeki uydunun pozisyon
vektörü
    #R        -Yermerkezli ekvatoral çerçevedeki gözlem noktasının
pozisyon vektörü
    #lstOS    -Gözlem noktasının yıldız zamanı (derece)
    #Lat      -Gözlem noktasının enlemi (derece)
    #r_sun    -Güneş'in yermerkezli ekvatoral çerçevedeki pozisyon
vektörü
    #aSun     -Güneş'in yükselme açısı (radyan)
    #Asun     -Güneş'in yerberi açısı (radyan)
    #a        -Uydunun yükselme açısı (radyan)
    #A        -Uydunun yerberi açısı (radyan)
    #distance -Uydu ile gözlem noktasının arasındaki mesafe

    #Güneş'in yükselişini hesaplanması:
                  #Yer-merkezli ekvatoral koordinat sisteminde gözlem
noktasına göre uydunun bağıl konum vektörü:
                    rho = r_S - R
                    print (rho)

    #Yer-merkezli ekvatoral koordinat sisteminden yüzey-merkezli ufuk
gözlem çerçevesine değiştirme matrisi:
                    th = lst*180/math.pi
                    ph = (Lat*180/math.pi)

                    Q_Xx = numpy.array([[-math.sin(th),
math.cos(th),          0],
                   [-math.sin(ph)*math.cos(th),  -
math.sin(ph)*math.sin(th),  math.cos(ph)],
                   [math.cos(ph)*math.cos(th),
math.cos(ph)*math.sin(th),  math.sin(ph)]])

    #Relativ pozisyon vektörünün yüzey-merkezli ufuk gözlem çerçevesine
göre yazılışı:
                    rhoTH =
numpy.linalg.inv(Q_Xx*numpy.linalg.inv(rho))

    #Relativ pozisyon vektörü yönündeki birim vektör:
                    rhoTHdir = rhoTH/numpy.linalg.norm(rho)

    #Uydunun, gözlem noktasına yerleştirilmiş yüzey-merkezli ufuk
gözlem çerçevesinde
    #yükseliş açısının bulunuşu:
                    aSun = math.asin(rhoTHdir[3])

    #Uydunun, gözlem noktasına yerleştirilmiş yüzey-merkezli ufuk
gözlem çerçevesinde
    #yerberi açısının bulunuşu:
```

```
                    A = math.acos(rhoTHdir[2]/math.cos(a))
                    if sign(sin(A)) == sign(rhoTHdir[1]/math.cos(a)):
                        A=A
                    else:
                        A = 2*math.pi - A
                #Bu fonksiyon uydunun ve güneşin ECI (Earth-Centered
inital/Dünya-Merkezli başlangıç)
    #pozisyon vektörlerini kullanarak Dünya'nın; uydunun ve Güneş'in
arasında, görüş alanında olup
    #olmadığını hesaplar.

                    RE = 6378
                    rsat = norm(r)
                    print(rsat)
                    rsun = norm(r_S)

    #Güneş'in ve uydunun pozisyon vektörlerinin arasındaki açı:
                    theta = math.acosd(dot(r_sat, r_sun)/rsat/rsun)

    #Angle between the satellite position vector and the radial to the
point
    #of tangency with the earth of a line from the satellite:
                    theta_sat = math.acosd(RE/rsat)

    #Angle between the sun position vector and the radial to the point
    #of tangency with the earth of a line from the sun:
                    theta_sun = math.acosd(RE/rsun)

    #Güneş'ten uyduya doğru olan bir çizginin Dünya ile keşistiğinin
bulunması:

                    if theta_sat + theta_sun <= theta:
                        light_switch = 0;   #evet
                    else:
                        light_switch = 1;   #hayır

                    if  nu == 1 and aSun*180/math.pi < -6:
                        v = 1
                    else:
                        v = 0

     #The visible pass data:
                    distance = numpy.linalg.norm(r-R)
                    data = [(a*180/math.pi), (A*180/math.pi), distance,
(aSun*180/math.pi)]
                    if v == 1 and visibleOld==0:
                    # Start of visible pass:
            #Output fonksiyonu, görünür geçiş çıktısını text dosyasına
yazar.

    #i      - Output(çıktı) tipi; start(başlangıç), maximum
altitude(maksimum yükseklik), end(son)
    #fileID -Çıktının yazıldığı dosyanın ismi
    #date   -Geçişin tarih ve zamanı
    #data   -Geçişin verisi
                        i=1
                        fileID = open('Visible_pass_output.txt','w+')
                        if i == 1:
```

```
                            fileID.write('Date of visible pass: {} {}
{} \n' .format(round(date[3]), round(date[2]), round(date[1])))
                            fileID.write('\n')
                            fileID.write('Start of the visible pass:\n
')
                            searchMax = 1
                            aMax = data[1]
                        elif v == 1 and visibleOld == 1:
                        #Finding maximum altitude for the visible pass:
                            a = data[1]
                        if aMax < a:
                                aMax = a;
                        elif searchMax == 1:
                            date = datevec(datenum(dateInstOld) + 1/8);
                            i=2
                            fileID =
open('Visible_pass_output.txt','w+')
                        elif i == 2:
                            fileID.write('Maximum altitude of the
visible pass:\n')

                        if i == 3:
                            fileID.write('Sun elevation:  {} \n'
.format(data[4], sep=":"))

                        else:
                            fileID.write('Sun elevation:  {} \n'
.format(data[4], sep=":"))
                            fileID.write('\n')

                            searchMax = 0;

                    elif v == 0 and visibleOld == 1:
#End of visible pass output:
                            fileID =
open('Visible_pass_output.txt','w+')
                            if i == 1:
                                    fileID.write('Date of visible pass:
{} {} {} \n' .format(round(date[3]), round(date[2]), round(date[1])))
                                    fileID.write('\n')
                                    fileID.write('Start of the visible
pass:\n ')
                            elif i == 2:
                                    fileID.write('Maximum altitude of
the visible pass:\n')
                            elif i == 3:
                                    fileID.write('End of the visible
pass:\n')

                                    fileID.write('Time:  {} {} {} \n'
.format(round(date[4]), round(date[5]), round(date[6]), sep=":"))
                                    fileID.write('Elevation:  {} \n'
.format(round(date[1]), sep=":"))
                                    fileID.write('Azimuth:  {} \n'
.format(round(date[2]), sep=":"))
                                    fileID.write('Azimuth Distance
(km):  {} \n' .format(round(date[3]), sep = ":"))
```

```
                            if i == 3:
                                    fileID.write('Sun elevation:  {}
\n' .format(data[4], sep=":"))

                            else:
                                    fileID.write('Sun elevation:  {}
\n' .format(data[4], sep=":"))
                                    fileID.write('\n')


                                    visibleOld = visible;
                                    dateInstOld = dateInst;
                                    dataOld = data;
                                    fileID.close()
TLE_to_COE_and_t0()
```