```
import numpy as np
from math import *
import matplotlib
# matplotlib.use('Qt5Agg')
import matplotlib.pyplot as plt
import sys
import re

#define constants
cAUD = 173.144632674
timeEpoch = 2457597.125
k = 0.01720209895
mu = 1.0
ecA = radians(23.434)
asteroidDes = '1994PN'

#load data
data = np.loadtxt(str(sys.argv[1]), dtype='bytes', skiprows=1).astype(str)
with open(str(sys.argv[1]), 'r') as f:
    asteroidDes = re.search('\((.*)\)', f.readline()).group(1).replace(" ","")

#sexagesimal to decimal
def sexToDecimal(ang, hparam):
    anglist = ang.split(":")
    for i in range(len(anglist)):
        anglist[i] = float(anglist[i])
    decimal = (anglist[1] / 60) + (anglist[2] / 3600)
    if anglist[0] < 0:
        ang = anglist[0] - decimal
    else:
        ang = anglist[0] + decimal
    if hparam == True:
        ang *= 15
    return ang

#decimal to sexagesimal
def decimalToSex(inp, rd):
    d = int(inp)
    inp = inp % 1
    m = str(int(inp * 60))
    m = m.zfill(2)
    s = round(((inp * 60) % 1) * 60, 2)
    if rd == "d":
        s = str(round(s, 1))
        s = s.zfill(4)
        if d > 0:
            d = "+" + str(d)
    if rd == "r":
        s = str(s).zfill(5)
    return "%s:%s:%s" % (d, m, s)

#Julian Date Functions
def getJo(date):
    joArr = date.split("-")
    for i in range(len(joArr)):
        joArr[i] = float(joArr[i])
    jodate = 367*joArr[0] - int((7*(joArr[0] + int((joArr[1]+9)/12)))/4) +
int(275 * joArr[1] / 9) + joArr[2] + 1721013.5
    return jodate

def getJD(date, time):
```

```
    UT = sexToDecimal(time, False)
    return date + (UT / 24)

#ecliptic rotation matrix
def eclRot(vec):
    ecRot = np.array([[1, 0, 0],
                      [0, cos(ecA), sin(ecA)],
                      [0, -sin(ecA), cos(ecA)]], dtype='float')
    vec = ecRot.dot(vec)
    return vec

#get orbital elements from state vectors function
def babyOD(r, rdot, t2, tE, param):
    ecRot = np.array([[1, 0, 0],
                      [0, cos(ecA), sin(ecA)],
                      [0, -sin(ecA), cos(ecA)]], dtype='float')
    if param == True:
        r = ecRot.dot(r)
        rdot = ecRot.dot(rdot)
    rmag = np.linalg.norm(r)
    hvec = np.cross(r, rdot)
    h = np.linalg.norm(hvec)
    rdotmag = np.linalg.norm(rdot)
    #semimajor axis
    a = ((2/rmag)-np.dot(rdot,rdot)/k**2)**-1
    #eccentricity
    e = (1-((h**2)/a/k/k))**0.5
    #inclination
    i = acos(hvec[2]/h)
    #longitude of ascending node
    osin = hvec[0]/(h*sin(i))
    ocos = ((-1)*hvec[1])/(h*sin(i))
    o = atan2(osin,ocos)
    #argument of perihelion
    sinwf = r[2]/(rmag*sin(i))
    coswf = (1/cos(o))*((r[0]/rmag)+(cos(i)*sinwf*sin(o)))
    wf = atan2(sinwf,coswf)
    cosf = (a*(1-e*e)/rmag - 1)/e
    sinf = a*(1-e*e)*np.dot(r,rdot)/h/e/rmag
    f = atan2(sinf,cosf)
    w = wf-f
    #mean anomaly
    nbod = k/sqrt(a**3)
    E = acos((1 - rmag/a)/e)
    if f < 0:
        E = 2*pi-E
    M2 = E - e*sin(E)
    Me = M2 + nbod*(tE-t2)
    return tE, a, e, degrees(i), degrees(o), degrees(w), degrees(Me)


#parse data
ndata = np.empty([4, 7], dtype='float')
for row in range(data.shape[0]):
    ndata[row,0] = float(getJD(getJo(data[row,0]),data[row,1]))
    ndata[row,2] = radians(sexToDecimal(data[row,2], True))
    ndata[row,3] = radians(sexToDecimal(data[row,3], False))
    ndata[row,4] = float(data[row,4])
    ndata[row,5] = float(data[row,5])
    ndata[row,6] = float(data[row,6])
```

```
    #define data structure
    timeArray = np.copy(ndata[:,0])
    raArray = np.copy(ndata[:,2])
    decArray = np.copy(ndata[:,3])
    R1 = np.array([ndata[0,4],ndata[0,5],ndata[0,6]])
    R2 = np.array([ndata[1,4],ndata[1,5],ndata[1,6]])
    R3 = np.array([ndata[2,4],ndata[2,5],ndata[2,6]])
    R4 = np.array([ndata[3,4],ndata[3,5],ndata[3,6]])


    #calculate time intervals
    tau = k*(timeArray[2]-timeArray[0])
    tau3 = k*(timeArray[2]-timeArray[1])
    tau1 = k*(timeArray[0]-timeArray[1])

    #calculate rho hats
    def hatRhos(rain, decin):
        return np.array([cos(rain)*cos(decin), sin(rain)*cos(decin), sin(decin)])
    rhohat1 = hatRhos(raArray[0], decArray[0])
    rhohat2 = hatRhos(raArray[1], decArray[1])
    rhohat3 = hatRhos(raArray[2], decArray[2])


    ##START TRUNCATED F+G SERIES FOR INITIAL GUESS (subsitutions galore)
    #scalar equations of range pt 1
    D0 = rhohat1.dot(np.cross(rhohat2, rhohat3))

    D11 = np.cross(R1, rhohat2).dot(rhohat3)
    D12 = np.cross(R2, rhohat2).dot(rhohat3)
    D13 = np.cross(R3, rhohat2).dot(rhohat3)

    D21 = np.cross(rhohat1, R1).dot(rhohat3)
    D22 = np.cross(rhohat1, R2).dot(rhohat3)
    D23 = np.cross(rhohat1, R3).dot(rhohat3)

    D31 = rhohat1.dot(np.cross(rhohat2, R1))
    D32 = rhohat1.dot(np.cross(rhohat2, R2))
    D33 = rhohat1.dot(np.cross(rhohat2, R3))

    A1 = tau3/tau
    A3 = -tau1/tau
    B1 = A1*(tau**2-tau3**2)/6
    B3 = A3*(tau**2-tau1**2)/6
    A = -((A1*D21)-D22+(A3*D23))/D0
    B = -((B1*D21)+(B3*D23))/D0
    E = -2*(rhohat2.dot(R2))
    F = np.linalg.norm(R2)**2

    #get roots and ask user
    roota = -(A**2 + A*E + F)
    rootb = -(2*A*B + B*E)
    rootc = -(B**2)

    coeffs=[rootc,0,0,rootb,0,0,roota,0,1]
    roots=np.polynomial.polynomial.polyroots(coeffs)
    realRoots = []

    for i in range(len(roots)):
        if np.imag(roots[i]) == 0 and np.real(roots[i]) > 0:
            realRoots.append(float(np.real(roots[i])))
    print(realRoots)
```

```
#matplotlib
r = np.arange(0.6, 1.7, 0.01)
plt.plot(r, r**8 + roota*r**6 + rootb*r**3 + rootc, '-')
plt.plot(r, r-r, '-')
plt.plot(realRoots, [0,0,0], 'bs')
plt.xlabel('|r| (AU)')
plt.title('Roots of |r| Polynomial')
plt.grid(True)
plt.show()

pick = eval(input("Pick a root to try by index: "))
rmag = realRoots[int(pick)]

u = 1/rmag**3

#get initial f+g
f1 = 1-0.5*u*tau1**2
g1 = tau1-(u*tau1**3)/6
f3 = 1-0.5*u*tau3**2
g3 = tau3-(u*tau3**3)/6

C2 = -1
C1 = g3/(f1*g3-f3*g1)
C3 = -g1/(f1*g3-f3*g1)

rho1 = abs((C1*D11+C2*D12+C3*D13)/(C1*D0))
rho2 = abs((C1*D21+C2*D22+C3*D23)/(C2*D0))
rho3 = abs((C1*D31+C2*D32+C3*D33)/(C3*D0))

#get initial position vectors
r1 = rhohat1 * rho1 - R1
r2 = rhohat2 * rho2 - R2
r3 = rhohat3 * rho3 - R3

#get initial velocity vector
d1 = -f3/(f1*g3-f3*g1)
d3 = f1/(f1*g3-f3*g1)
r2dot = d1*r1 + d3*r3

#iteration control variables
tolerance = 1e-12
rOld = 0
counter = 0
tau1c = tau1
tau3c = tau3
tauc = tau

#MAIN ITERATION
print("""
-------------------------------------------------------------------------
Beginning Iteration
Tolerance: %s
-------------------------------------------------------------------------""" %
(tolerance))
while abs(rOld-rmag) > tolerance:
    counter += 1

    #closed form f+g functions
    a = ((2/rmag)-np.dot(r2dot,r2dot))**-1
    n = 1/sqrt(a**3)
```

```
        sub1 = 1-rmag/a
        sub2 = rmag*np.linalg.norm(r2dot)/n/a/a

        #newtons method to find delta E
        x1 = n*tau1c
        x1old = 0
        while abs(x1-x1old)>0.0000000001:
            x1old = x1
            fx1 = x1old-sub1*sin(x1old)+sub2*(1-cos(x1old))-n*tau1c
            fprimex1 = 1-sub1*cos(x1old)+sub2*sin(x1old)
            x1 = x1old-fx1/fprimex1
        x3 = n*tau3c
        x3old = 0
        while abs(x3-x3old)>0.0000000001:
            x3old = x3
            fx3 = x3old-sub1*sin(x3old)+sub2*(1-cos(x3old))-n*tau3c
            fprimex3 = 1-sub1*cos(x3old)+sub2*sin(x3old)
            x3 = x3old-fx3/fprimex3

        #calculte new f+g
        f1 = 1-a*(1-cos(x1))/rmag
        f3 = 1-a*(1-cos(x3))/rmag
        g1 = tau1c+(sin(x1)-x1)/n
        g3 = tau3c+(sin(x3)-x3)/n

        #truncated f+g taylor series
        #u = 1/rmag**3
        #z = r2.dot(r2dot)/rmag/rmag
        #q = r2dot.dot(r2dot)/rmag/rmag-u
        #f1 = 1-

        C1 = g3/(f1*g3-f3*g1)
        C3 = -g1/(f1*g3-f3*g1)
        d1 = -f3/(f1*g3-f3*g1)
        d3 = f1/(f1*g3-f3*g1)

        rho1 = abs((C1*D11+C2*D12+C3*D13)/(C1*D0))
        rho2 = abs((C1*D21+C2*D22+C3*D23)/(C2*D0))
        rho3 = abs((C1*D31+C2*D32+C3*D33)/(C3*D0))

        #get new state vectors
        r1 = rhohat1 * rho1 - R1
        r2 = rhohat2 * rho2 - R2
        r3 = rhohat3 * rho3 - R3
        r2dot = d1*r1 + d3*r3

        rOld = rmag + 0
        rmag = np.linalg.norm(r2)

        #debug
        print("Iteration %s: delta r = %s" % (counter, abs(rmag-rOld)))

        #light time correction
        tauc = k*((timeArray[2]-rho3/cAUD)-(timeArray[0]-rho1/cAUD))
        tau3c = k*((timeArray[2]-rho3/cAUD)-(timeArray[1]-rho2/cAUD))
        tau1c = k*((timeArray[0]-rho1/cAUD)-(timeArray[1]-rho2/cAUD))

    ##output
    r2dot = k*r2dot
    orbEl = babyOD(r2, r2dot, timeArray[1], timeEpoch, True)
    print()
```

```
    str1 =
    """--------------------------------------------------------------------------
    Initial Orbital Determination Results for %s
    Calculations by Alexander Davenport
    --------------------------------------------------------------------------
    Iterations Needed: %s
    Time of Central Observation: JD %s

    r vector: (equatorial, AU)
    %s

    rdot vector: (equatorial, AU/day)
    %s

    r vector: (ecliptic, AU)
    %s

    rdot vector: (ecliptic, AU/day)
    %s

    range: (AU)
    %s

    """ % (asteroidDes, counter, timeArray[1], r2, r2dot, eclRot(r2), eclRot(r2dot),
    rho2)

    str2 = """Orbital Elements (Epoch JD %s):
    a: %s
    e: %s
    i: %s
    O: %s
    w: %s
    Me: %s""" % orbEl #tE, a, e, degrees(i), degrees(o), degrees(w), degrees(Me)

    output = str1 + str2
    print(output)
    print("--------------------------------------------------------------------------
    -\n")
    printOpt = input("Do you want to save results to file? (y/n): ").lower()
    if printOpt == "y":
        f = open("%s_initialODresults.txt" % asteroidDes, "w")
        f.write(output)
        f.close()
        print('File Created!: %s_initialODresults.txt' % asteroidDes)
    print("\
    n--------------------------------------------------------------------------")
    input("Press 'Enter' to exit")
```