find_closest_approach.py

November 12, 2017

In [1]:
```
'''
Finds the time of closest approach between Mars and Earth. Prints out the closest appro
Distance is printed out for verification purposes. Time is used in the simulation prog
'''
import math
from math import sin, cos

class Planet:
    '''
    Instantiates a planet object for easy calculation and tracking of celestial orbits
    '''
    def __init__(self, semimajor_axis, eccentricity, inclination, mean_longitude, l_per
        '''
        All parameters given in Keplerian Elements by E M Standish. Each parameter is g
        and the second value being the rate of change in the value with respect to tim
        The primary units for each non-self paramater are as follows: au, none, deg, d
        J2000.0.
        '''
        self.semimajor_axis = semimajor_axis
        self.eccentricity = eccentricity
        self.inclination = inclination
        self.l_ascending = l_ascending
        self.l_perihelion = l_perihelion
        self.mean_longitude = mean_longitude

    def GetLocation(self, days):
        '''
        Follows algorithm given on pages 1,2  of Keplerian Elements by E M Standish. Sp
        Days is Julian Days
        '''
        time = self.NormalizeTime(days)
        a = (self.semimajor_axis[0] + self.semimajor_axis[1]*time)*149597870.7 # conve
        e = self.eccentricity[0] + self.eccentricity[1]*time
        I = self.inclination[0] + self.inclination[1]*time
        L = self.mean_longitude[0] + self.mean_longitude[1]*time
        ohm_bar = self.l_perihelion[0] + self.l_perihelion[1]*time
        OHM = self.l_ascending[0] + self.l_ascending[1]*time
```

```
        ohm = ohm_bar - OHM
        M = L - ohm_bar
        E = math.radians(self.GetEccentricAnomaly(M, e))
        I = math.radians(I)
        ohm = math.radians(ohm)
        OHM = math.radians(OHM)

        x_prime = a*(cos(E) - e)
        y_prime = a*math.sqrt(1-e**2)*sin(E)
        x = (cos(ohm)*cos(OHM)-sin(ohm)*sin(OHM)*cos(I))*x_prime
        x = x + (-sin(ohm)*cos(OHM) - cos(ohm)*sin(OHM)*cos(I))*y_prime
        y = (cos(ohm)*cos(OHM)+sin(ohm)*cos(OHM)*cos(I))*x_prime
        y = y + (-sin(ohm)*sin(OHM) + cos(ohm)*cos(OHM)*cos(I))*y_prime
        z = sin(ohm)*sin(I)*x_prime + cos(ohm)*sin(I)*y_prime
        return (x, y, z)

    def NormalizeTime(self, days):
        '''
        Since all time related units in __init__ are given by unit/centuries, this con
        a usable centuries since J2000.0 unit
        '''
        return (days - 2451545.0)/36525

    def GetEccentricAnomaly(self, M, e):
        '''
        Algorithm given by equations 8-36 and 8-37 of Keplerian Elements by E M Standi
        '''
        if M > 180:
            M -= 360
        e_star = e
        e = math.radians(e)
        incr = 1
        E = M + e_star*sin(M)
        dE = 1
        tol = 1e-6
        while abs(dE) > tol:
            dM = M - (E - e_star*sin(E))
            dE = dM/(1-e*cos(E))
            E = E + dE

        return E

def lcm(n1, n2):
    '''
    Calculates least common multiple in case that ends up being useful
    '''
    return int((n1 * n2)/math.gcd(n1,n2))
```

```python
def distance(co1, co2):
    '''
    Returns the scalar distance between two points/vectors
    '''
    temp = 0
    for i in range(len(co1)):
        temp += (co1[i] - co2[i])**2
    return math.sqrt(temp)

earth = Planet((1.00000261, 0.00000562), (0.01671123, -0.00004392), (-0.00001531, -0.0
               (100.46457166, 35999.37244981), (102.93768193, 0.32327364), (0.0, 0.0))

mars = Planet((1.52371034, 0.00001847), (0.09339410, 0.00007882), (1.84969142, -0.0081
              (-4.55343205, 19140.30268499), (-23.94362959, 0.44441088),
              (49.55953891, -0.29257343)) # Values from Table 1 of Keplerian Elements

loc_earth = []
loc_mars = []
earth_orbital_period = 36525
mars_orbital_period = 687

best = [9e99, 0] # Distance (km), time (days)

for i in range(245154500, 245154500+earth_orbital_period*100, 1):
    t = i/100
    loc_earth = earth.GetLocation(t)
    loc_mars = mars.GetLocation(t)
    dist = distance(loc_earth, loc_mars)
    if dist < best[0]:
        best = [dist, t]

print(best)
```

```
[58214181.23314717, 2487971.95]
```