HW 4 - Tory Smith

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint, solve_ivp
import pandas as pd
from datetime import datetime, timedelta
```

AU-76/FK5 reduction of position from ECEF (ITRF) to ECI (ICRF) NOTE: This method uses the IAU-1976 Precession Model & IAU-1980 Theory of Nutation

For this problem, I used the method as described in Vallado (5th Edition).

First, I define the Rotation Matrices Below

In [ ]:
```
#Rotation Martices
def R1(theta):
    return np.array([[1, 0, 0], \
                     [0, np.cos(theta), np.sin(theta)], \
                     [0, -np.sin(theta), np.cos(theta)]])
def R2(theta):
    return np.array([[np.cos(theta), 0, -np.sin(theta)],\
                    [0, 1, 0], \
                    [np.sin(theta), 0, np.cos(theta)]])
def R3(theta):
    return np.array([[np.cos(theta), np.sin(theta), 0], \
                     [-np.sin(theta), np.cos(theta), 0], \
                    [0, 0, 1]])
```

Then, using the UTC Julian date provided, $\Delta UT1$ from Earth Orientation Parameters (EOP) file and Leap Seconds from Bulletin C, I calculate T_{UT1} and T_{TT}. Using the Earth Rotation Angles from the EOP file, I formulate the Polar Motion Matrix (W). I then find GMST using T_{UT1} and add this to the Equation of the Equinoxes using the mean nutation in obliquity $\bar{\epsilon}_m$ the nutation in longitude $\Delta\psi$ and nutation in obliquity $\Delta\epsilon$ with the anamolies provided in the nut80.dat file to find GAST. GAST is then used to find the Sidereal Rotation Matrix (R). $\bar{\epsilon}_m$, $\Delta\psi$ and $\Delta\epsilon$ are then used to find the Nutation Matrix (N). The Precession Matrix (P) is foudn using ζ_a, θ_a and z_a.

Finally, r_{ECI} is calculated by the formula:

r_{ECI} = $[P][N][R][W]r_{ECEF}$

where,

r_{ECEF} is given as: $[-28738.3218400000, -30844.0723200000, -6.71800000000000][km]$

In my algorithm r_{ECI} is found as:

r_{ECI} = $[19165.44514777874, -37549.06140374086, -41.043609948282580][km]$

with an error of $[0.00110197531, 0.00056247, -0.00001041][km]$ from the provided solution.

In [ ]:
```
#convert from ECEF to ECI Vallado

# time constants
JD2000 = 2451545.0
# JD_UTC = 2453101.82741
JD_UTC = 2458088.50055556
```

```
#T_UT1
del_UT1 = 248.5001E-3
JD_UT1 = JD_UTC + del_UT1/86400
T_UT1 = (JD_UT1-JD2000)/36525

#T_TT
TAI = JD_UTC + 37/86400
JD_TT = TAI + 32.184/86400
T_TT = (JD_TT-JD2000)/36525

#radians conversions
arc_sec_to_rad = np.pi/(180*3600)
deg2rad = np.pi/180

#ECEF coordinates
r_ECEF = np.array([-28738.3218400000, -30844.0723200000, -6.71800000000000])

#Earth Rotation Angles
x_p = 124.136E-3*arc_sec_to_rad
y_p = 236.728E-3*arc_sec_to_rad

# Polar Motion Matrix
W = np.matmul(R1(y_p), R2(x_p))
# r_PEF = np.matmul(W, r_ECEF)

#Greenwich Mean Sidereal Time
GMST = 67310.54841 + (876600*3600 + 8640184.812866)*T_UT1 + 0.093104*T_UT1**2 - 6.2E-6*T_U

#convert GMST to radians
GMST = GMST/240*deg2rad

#anamolies
r = 360
Mmoon = (134.96298139 + (1325*r + 198.8673981)*T_TT + 0.0086972*T_TT**2 + 1.78E-5*T_TT**3)
Mdot = (357.52772333 + (99*r + 359.0503400)*T_TT - 0.0001603*T_TT**2 - 3.3E-6*T_TT**3)
uMoon = (93.27191028 + (1342*r + 82.0175381)*T_TT - 0.0036825*T_TT**2 + 3.1E-6*T_TT**3)
Ddot = (297.85036306 + (1236*r + 307.1114800)*T_TT - 0.0019142*T_TT**2 + 5.3E-6*T_TT**3)
lamMoon = (125.04452222 - (5*r + 134.1362608)*T_TT + 0.0020708*T_TT**2 + 2.2E-6*T_TT**3)
alpha = np.array([Mmoon, Mdot, uMoon, Ddot, lamMoon])*deg2rad

# IAU1980 Theory of Nutation model
dat_file = "nut80.dat"

#nutaton model column names
column_names = ['ki1', 'ki2', 'ki3', 'ki4', 'ki5', 'Aj', 'Bj', 'Cj', 'Dj', 'j']

#nutation dataframe
df = pd.read_csv(dat_file, sep="\s+", names=column_names)

#nutation in longitude
del_psi = np.dot((df['Aj']*10**-4 + df['Bj']*10**-4*T_TT)*arc_sec_to_rad, np.sin(np.dot(df

#nutation in obliquity
del_epsilon = np.dot((df['Cj']*10**-4 + df['Dj']*10**-4*T_TT)*arc_sec_to_rad, np.cos(np.do

#mean obliquity of the ecliptic
epsilon_m = 84381.448 - 46.8150*T_TT - 0.00059*T_TT**2 + 0.001813*T_TT**3

#EOP corrections
# ddel_psi = -104.524E-3
# ddel_epsilon = -8.685E-3

#conversion to radians
epsilon_m = epsilon_m*arc_sec_to_rad

#true obliquity of the ecliptic
epsilon = epsilon_m + del_epsilon
```

```
#equation of the equinoxes
Eq_eq = del_psi*np.cos(epsilon_m) + 0.000063*arc_sec_to_rad*np.sin(2*alpha[4]) + 0.00264*a

#greenwich apparent sidereal time
GAST = GMST + Eq_eq

#sidereal rotation matrix
R = R3(-GAST)
# r_TOD = np.matmul(R, r_PEF)

#nutation matrix R1, R3, R1
N = np.matmul(np.matmul(R1(-epsilon_m), R3(del_psi)), R1(epsilon))
# r_mod = np.matmul(N, r_TOD)

#precession angles
C_a = (2306.2181*T_TT + 0.30188*T_TT**2 + 0.017998*T_TT**3)*arc_sec_to_rad
theta_a = (2004.3109*T_TT - 0.42665*T_TT**2 - 0.041833*T_TT**3)*arc_sec_to_rad
z_a = (2306.2181*T_TT + 1.09468*T_TT**2 + 0.018203*T_TT**3)*arc_sec_to_rad

#precession matrix
P = np.matmul(np.matmul(R3(C_a), R2(-theta_a)), R3(z_a))

r_ECI = np.matmul(np.matmul(np.matmul(np.matmul(P, N), R), W), r_ECEF)


print(f'Result: {r_ECI[0]:.8f} {r_ECI[1]:.8f} {r_ECI[2]:.8f} [km]')
solution = np.array([19165.44514777874, -37549.06140374086, -41.043609948282580])
print(f'Error: {(r_ECI[0]-solution[0]):.8f} {r_ECI[1]-solution[1]:.8f} {r_ECI[2]-solution[
```

```
Result: 19165.44624975 -37549.06084127 -41.04362036 [km]
Error: 0.00110198 0.00056247 -0.00001041 [km]
```