Satellite Data Retreval

Jay Dickson | S3719855

Question 1

Coordinate Frame (1pt)

All ephemeris formats must be in the mean equator/mean equinox (MEME) J2000.0 frame with positional values (x, y, z) given in kilometer units and velocity values (dX, dY, dZ) given in units of kilometers/second. **(Extract from Handbook)**

Frame Type (1pt)

Frame is inertial

Question 2

Load Data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# Define the column names
column_names = ['YEAR', 'YEARDAY', 'X', 'Y', 'Z', 'Vx', 'Vy', 'Vz']

# Read the CSV file
ephemeris_data = pd.read_csv('starlink_46753_student_69.csv', header=None,
names=column_names)

print(ephemeris_data.head())
```

```
   YEAR     YEARDAY            X           Y            Z        Vx        Vy  \
0  2023  149.484514 -6181.567780  384.549383 -3107.145551 -2.823489 -4.937861
1  2023  149.485208 -6337.528267   87.661254 -2799.700605 -2.373292 -4.954849
2  2023  149.485903 -6466.154651 -209.605137 -2480.147020 -1.912685 -4.950469
3  2023  149.486597 -6566.881918 -505.967654 -2149.862206 -1.443654 -4.924726
4  2023  149.487292 -6639.265038 -800.146752 -1810.271830 -0.968221 -4.877712

         Vz
0  5.015733
1  5.228743
2  5.419218
3  5.586314
4  5.729288
```

Find timestep (1pt)

```
# Calculate the time difference between each row in seconds
ephemeris_data['time_diff_seconds'] = ephemeris_data['YEARDAY'].diff() * 86400
print(ephemeris_data['time_diff_seconds'][1])
```

```
60.00000000158252
```

Find Change in Velocity $||\dot{R}||$

```
# Calculate the differences in position
ephemeris_data['dX'] = ephemeris_data['X'].diff()
ephemeris_data['dY'] = ephemeris_data['Y'].diff()
ephemeris_data['dZ'] = ephemeris_data['Z'].diff()

# Calculate the magnitude of the position difference for each row
ephemeris_data['pos_diff'] = np.sqrt(ephemeris_data['dX']**2 + ephemeris_data['dY']**2 +
ephemeris_data['dZ']**2)

# Calculate the velocity in kilometers per second as the position difference over time
difference
ephemeris_data['velocity_kmps'] = ephemeris_data['pos_diff'] /
ephemeris_data['time_diff_seconds']

R_dot = ephemeris_data['velocity_kmps']
```

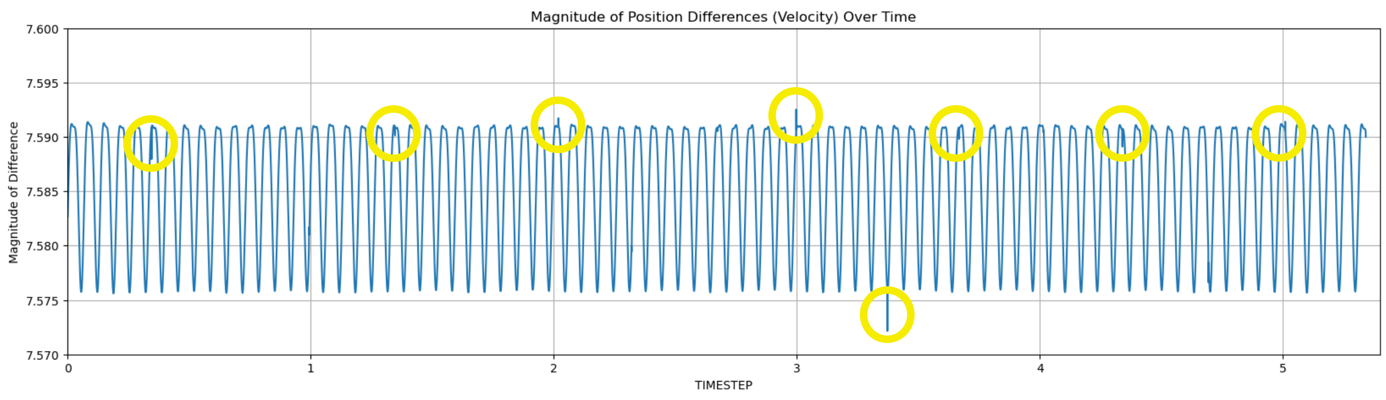
Find change in acceleration $||\ddot{R}||$

```
# Calculate the differences in position
ephemeris_data['ddX'] = ephemeris_data['dX'].diff()
ephemeris_data['ddY'] = ephemeris_data['dY'].diff()
ephemeris_data['ddZ'] = ephemeris_data['dZ'].diff()

# Calculate the magnitude of the position difference for each row
ephemeris_data['vel_diff'] = np.sqrt(ephemeris_data['ddX']**2 + ephemeris_data['ddY']**2
+ ephemeris_data['ddZ']**2)

# Now, we need to calculate the time difference between each row in seconds
ephemeris_data['time_diff_seconds'] = ephemeris_data['YEARDAY'].diff() * 86400

# Calculate the velocity in kilometers per second as the position difference over time
difference
ephemeris_data['acc_mpsps'] = (ephemeris_data['vel_diff'] /
(ephemeris_data['time_diff_seconds']**2))* 1000

R_double_dot = ephemeris_data['acc_mpsps']
```

Plot Velocity Data (4pts)

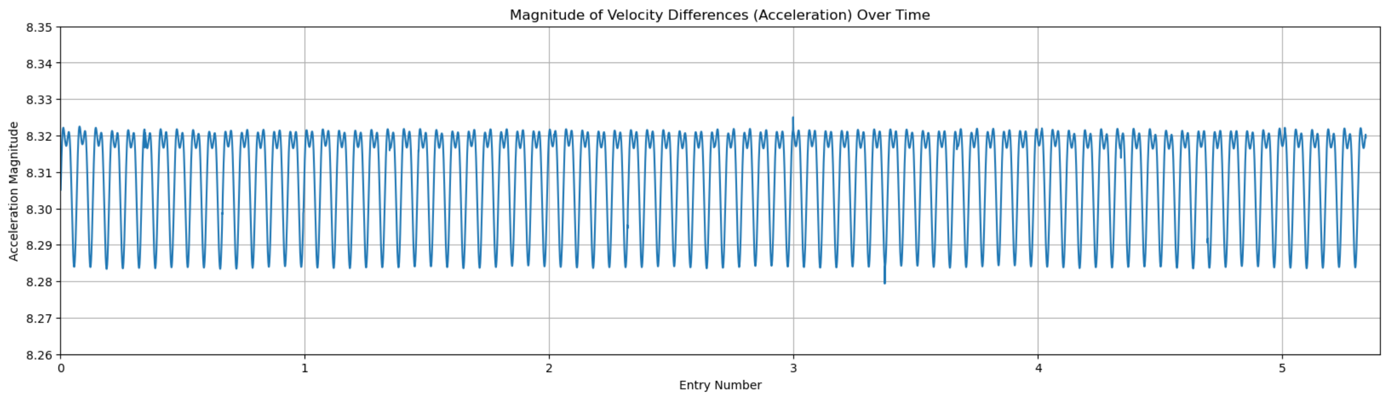
```
days = ephemeris_data['YEARDAY'] - ephemeris_data['YEARDAY'][0]
```

```
plt.figure(figsize=(20, 5))
plt.plot(days, R_dot)
plt.title('Magnitude of Position Differences (Velocity) Over Time')
plt.xlabel('TIMESTEP')
plt.ylabel('Magnitude of Difference')
plt.xlim(0, 5.4)
plt.ylim(7.57, 7.6)
plt.grid(True)
plt.show()
```



Plot Acceleration Data (4pts)

```
plt.figure(figsize=(20, 5))
plt.plot(days, R_double_dot)
plt.title('Magnitude of Velocity Differences (Acceleration) Over Time')
plt.xlabel('Entry Number')
plt.ylabel('Acceleration Magnitude')
plt.xlim(0, 5.4)
plt.ylim(8.26, 8.35)
plt.grid(True)
plt.show()
```



Identify Discontinuities

Discontinuities are highlighted on the above graph.

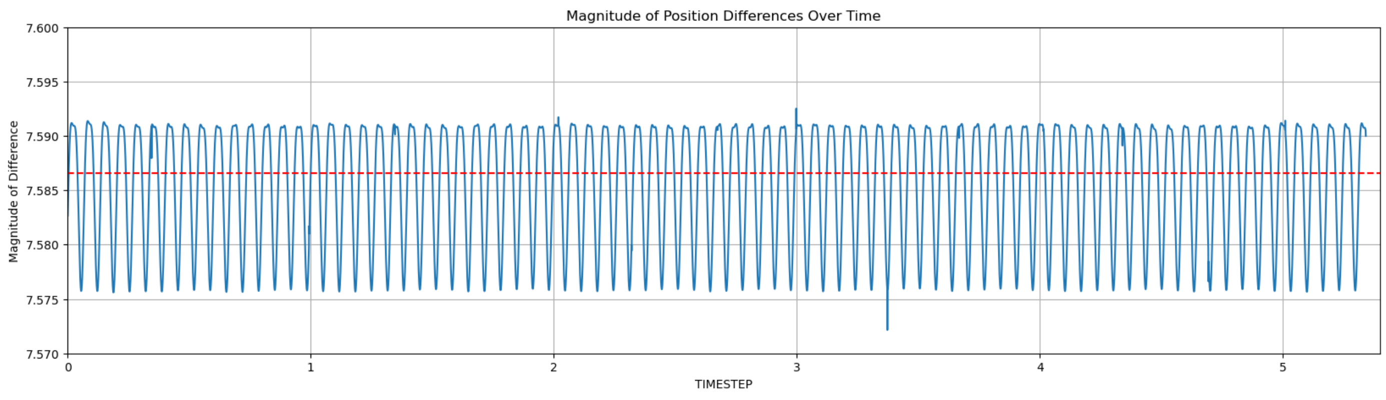
Approimate acceleration and velocity

```
# Constants
G = 6.67430e-20  # gravitational constant, km^3 kg^-1 s^-2
M_earth = 5.972e24  # mass of the Earth, kg
R_earth = 6371  # radius of the Earth, km

# Calculate the average altitude
average_radius = np.mean(np.sqrt(ephemeris_data['X']**2 + ephemeris_data['Y']**2 +
ephemeris_data['Z']**2))
average_altitude = average_radius - R_earth

# Calculate gravitational acceleration at average altitude
g_h = G * M_earth / (average_radius**2)

# Calculate orbital velocity for a circular orbit at average altitude
v_circular = np.sqrt(G * M_earth / average_radius)

# Output the results
print(f"Average Altitude: {average_altitude} km")
print(f"Gravitational Acceleration at this altitude: {g_h} km/s^2")
print(f"Orbital Velocity for a circular orbit at this altitude: {v_circular} km/s")
```

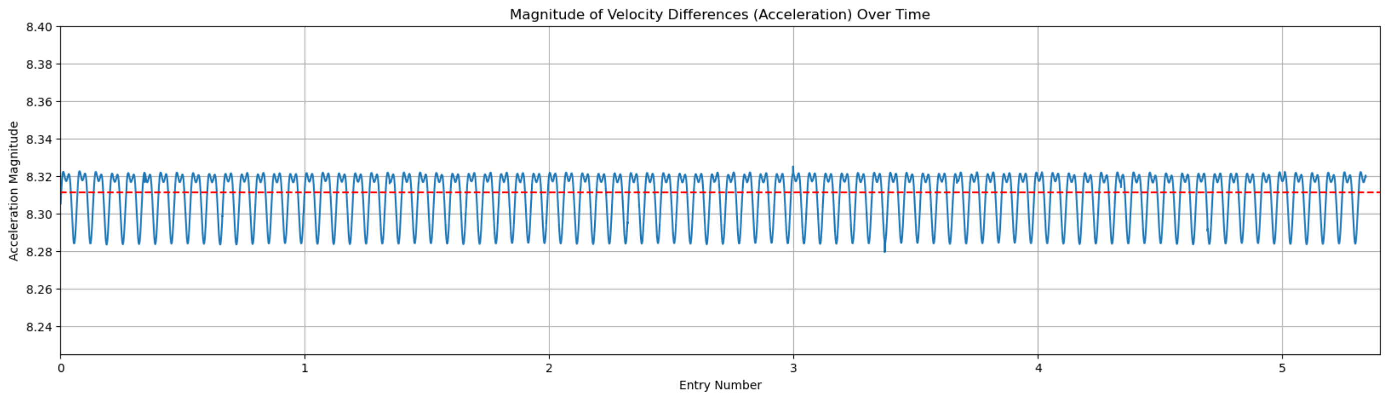
```
Average Altitude: 554.2261378693547 km
Gravitational Acceleration at this altitude: 0.008311082697495855 km/s^2
Orbital Velocity for a circular orbit at this altitude: 7.5865754547550655 km/s
```

Compare with plots

```
plt.figure(figsize=(20, 5))
plt.plot(days, R_dot)
plt.axhline(y=v_circular, color='r', linestyle='--', label='Expected Orbital Velocity
(v_circular)')
plt.xlim(0, 5.4)
plt.ylim(7.57, 7.6)
plt.title('Magnitude of Position Differences Over Time')
plt.xlabel('TIMESTEP')
plt.ylabel('Magnitude of Difference')
plt.grid(True)
plt.show()
```



```python
plt.figure(figsize=(20, 5))
plt.plot(days, R_double_dot)
plt.title('Magnitude of Velocity Differences (Acceleration) Over Time')
plt.axhline(y=(g_h * 1000), color='r', linestyle='--', label='Expected Gravitational
Acceleration (g_h')
plt.xlim(0, 5.4)
plt.ylim(8.225, 8.400)
plt.xlabel('Entry Number')
plt.ylabel('Acceleration Magnitude')
plt.grid(True)
plt.show()
```



Does it match up?

The estimated data lines up well with the ephemeris data. The value occilates around the expected value for both plots. The variations could be due to the orbit not being perfectly circular leading to slight periodic variations.

Question 3

Extract 3 TLEs (3pts)

TLE Data

- 1 46753U 20074Q   23149.42449537 .00008907 00000-0 61574-3 0 9995
- 2 46753 53.0530 199.3189 0001490 76.9720 283.1435 15.06421869143655

- 1 46753U 20074Q   23149.15911209 .00008886 00000-0 61440-3 0 9993
- 2 46753 53.0528 200.5100 0001497 77.4825 282.6332 15.06417404143637

- 1 46753U 20074Q   23148.42930437 .00006219 00000-0 43603-3 0 9995
- 2 46753 53.0527 203.7866 0001488 76.5916 283.5239 15.06401814143501

Question 4

Use SGP4 to predict positions for each TLE at each timestep

```
from sgp4.api import Satrec, WGS72
from sgp4.api import jday
from datetime import datetime, timedelta
from astropy.coordinates import TEME, CartesianDifferential, CartesianRepresentation
from astropy.coordinates import GCRS, EarthLocation
from astropy.time import Time
import astropy.units as u
```

Define a function that uses SGP4 and Astropy to convert the coordinate frame and find the position for the TLE at each timestamp in the Ephemeris data.

```
def getSatellitePosistion(tle1, tle2, year, yearday):

    # Initialize a satellite record from the TLE lines
    satellite = Satrec.twoline2rv(tle1, tle2)

    # Convert year to an integer
    year_int = int(year)

    # Extract the whole day and the fractional day parts
    day_of_year_int = int(yearday)  # Gets the whole day part
    fraction_of_day = yearday - day_of_year_int  # Gets the fractional part of the day

    # Calculate the exact time that the fractional day represents
    total_seconds_in_day = 24 * 60 * 60
    seconds_of_day = fraction_of_day * total_seconds_in_day
    hours = int(seconds_of_day // (60 * 60))
    minutes = int((seconds_of_day % (60 * 60)) // 60)
    seconds = seconds_of_day % 60

    # Create the datetime object
    start_of_year = datetime(year_int, 1, 1)
    delta_days = timedelta(days=day_of_year_int - 1)
    time_of_day = timedelta(hours=hours, minutes=minutes, seconds=seconds)
    date_time = start_of_year + delta_days + time_of_day

    # Extract the individual date and time components
    year = date_time.year
    month = date_time.month
    day = date_time.day
    hour = date_time.hour
    minute = date_time.minute
    second = date_time.second
    microsecond = date_time.microsecond

    # Convert the date and time to Julian Date for SGP4
    jd, fr = jday(year, month, day, hour, minute, second + microsecond * 1e-6)

    # Use SGP4 to find the position and velocity
    e, r_TEME, v_TEME = satellite.sgp4(jd, fr)

    if e == 0:
        # Create Astropy TEME coordinates with velocities
        teme_p = CartesianRepresentation(r_TEME*u.km,
differentials=CartesianDifferential(v_TEME*u.km/u.s))
        teme_v = TEME(teme_p, obstime=Time(date_time))

        # Convert TEME to GCRS (a good approximation of J2000)
        gcrs = teme_v.transform_to(GCRS(obstime=Time(date_time)))

        return(gcrs.cartesian.xyz)

    else:
        return(e)
```

```
#TLE1
tle1_line1 = '1 46753U 20074Q   23149.42449537  .00008907  00000-0  61574-3 0  9995'
tle1_line2 = '2 46753  53.0530 199.3189 0001490  76.9720 283.1435 15.06421869143655'

#TLE2
tle2_line1 = '1 46753U 20074Q   23149.15911209  .00008886  00000-0  61440-3 0  9993'
tle2_line2 = '2 46753  53.0528 200.5100 0001497  77.4825 282.6332 15.06417404143637'

#TLE3
tle3_line1 = '1 46753U 20074Q   23148.42930437  .00006219  00000-0  43603-3 0  9995'
tle3_line2 = '2 46753  53.0527 203.7866 0001488  76.5916 283.5239 15.06401814143501'
```

Find all satellite positions

```
ephemeris_data['tle1_position'] = ephemeris_data.apply(lambda row:
getSatellitePosistion(tle1_line1, tle1_line2, row["YEAR"], row["YEARDAY"]), axis=1)
```

```
ephemeris_data['tle2_position'] = ephemeris_data.apply(lambda row:
getSatellitePosistion(tle2_line1, tle2_line2, row["YEAR"], row["YEARDAY"]), axis=1)
```

```
ephemeris_data['tle3_position'] = ephemeris_data.apply(lambda row:
getSatellitePosistion(tle3_line1, tle3_line2, row["YEAR"], row["YEARDAY"]), axis=1)
```

```
#Remove units from the data
ephemeris_data['tle1_position'] = ephemeris_data['tle1_position'].apply(lambda pos:
(pos[0].to(u.km).value, pos[1].to(u.km).value, pos[2].to(u.km).value))
ephemeris_data['tle2_position'] = ephemeris_data['tle2_position'].apply(lambda pos:
(pos[0].to(u.km).value, pos[1].to(u.km).value, pos[2].to(u.km).value))
ephemeris_data['tle3_position'] = ephemeris_data['tle3_position'].apply(lambda pos:
(pos[0].to(u.km).value, pos[1].to(u.km).value, pos[2].to(u.km).value))
```

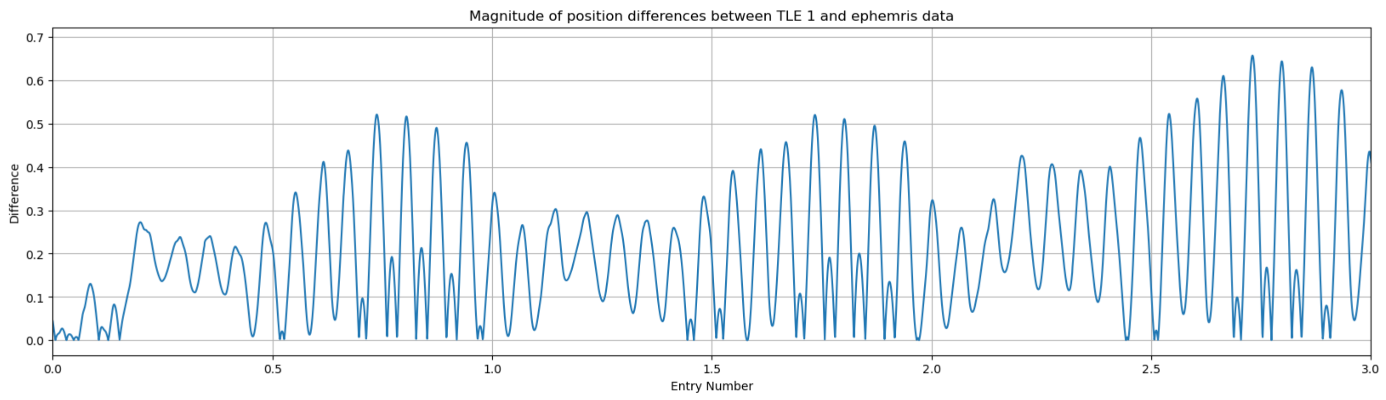
Calulate the magnitudes and differences between TLE positions and Ephemris Positions

```
# Calculate the magnitude of the position for each row
ephemeris_data['posMagnitude'] = abs(np.sqrt(ephemeris_data['X']**2 +
ephemeris_data['Y']**2 + ephemeris_data['Z']**2))


# Calculate the magnitude for the TLE positions
ephemeris_data['tle1_Posistion_Magnitude'] = ephemeris_data['tle1_position'].apply(lambda
pos: abs(np.linalg.norm(np.array([pos[0], pos[1], pos[2]]))))
ephemeris_data['tle2_Posistion_Magnitude'] = ephemeris_data['tle2_position'].apply(lambda
pos: abs(np.linalg.norm(np.array([pos[0], pos[1], pos[2]]))))
ephemeris_data['tle3_Posistion_Magnitude'] = ephemeris_data['tle3_position'].apply(lambda
pos: abs(np.linalg.norm(np.array([pos[0], pos[1], pos[2]]))))
```

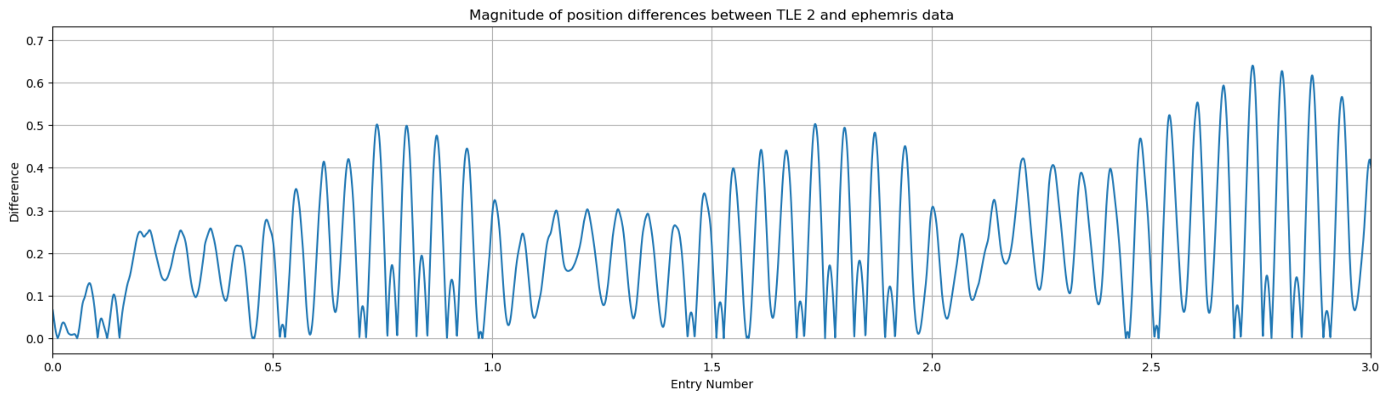
```
# Find differences between TLE pos and ephemris pos
ephemeris_data['tle1_PosDifference'] = ephemeris_data['tle1_Posistion_Magnitude'] -
ephemeris_data['posMagnitude']
ephemeris_data['tle2_PosDifference'] = ephemeris_data['tle2_Posistion_Magnitude'] -
ephemeris_data['posMagnitude']
ephemeris_data['tle3_PosDifference'] = ephemeris_data['tle3_Posistion_Magnitude'] -
ephemeris_data['posMagnitude']
```

Plot magnitude of differences between TLE and Ephemris Posistions (15 pts)

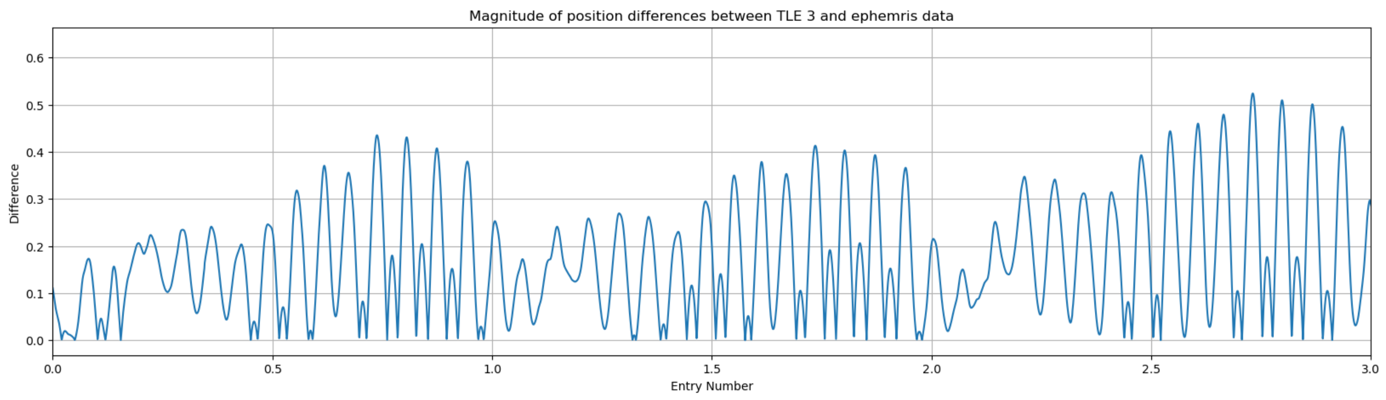
```
#Tle 1
plt.figure(figsize=(20, 5))
plt.plot(days, abs(ephemeris_data['tle1_PosDifference']))
plt.title('Magnitude of position differences between TLE 1 and ephemris data')
plt.xlim(0, 3)
plt.xlabel('Entry Number')
plt.ylabel('Difference')
plt.grid(True)
plt.show()
```



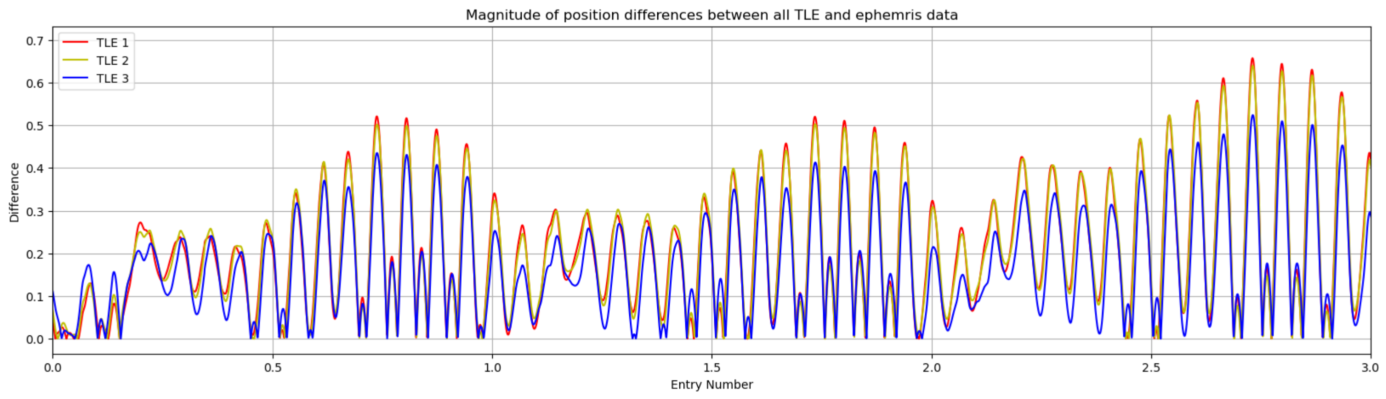
```
#Tle 2
plt.figure(figsize=(20, 5))
plt.plot(days, abs(ephemeris_data['tle2_PosDifference']))
plt.title('Magnitude of position differences between TLE 2 and ephemris data')
plt.xlim(0, 3)
plt.xlabel('Entry Number')
plt.ylabel('Difference')
plt.grid(True)
plt.show()
```



```
#Tle 3
plt.figure(figsize=(20, 5))
plt.plot(days, abs(ephemeris_data['tle3_PosDifference']))
plt.title('Magnitude of position differences between TLE 3 and ephemris data')
plt.xlim(0, 3)
plt.xlabel('Entry Number')
plt.ylabel('Difference')
plt.grid(True)
plt.show()
```



```
#Combined Graph
plt.figure(figsize=(20, 5))
plt.plot(days, abs(ephemeris_data['tle1_PosDifference']), color='r', linestyle='-',
label='TLE 1')
plt.plot(days, abs(ephemeris_data['tle2_PosDifference']), color='y', linestyle='-',
label='TLE 2')
plt.plot(days, abs(ephemeris_data['tle3_PosDifference']), color='b', linestyle='-',
label='TLE 3')
plt.title('Magnitude of position differences between all TLE and ephemris data')
plt.xlim(0, 3)
plt.xlabel('Entry Number')
plt.ylabel('Difference')
plt.legend()
plt.grid(True)
plt.show()
```



Question 5

Identify Basis Vectors (3pts)

- $\hat{W}$ - Cross Track
- $\hat{R}$ - Radial
- $\hat{S}$ - Along Track

Find Transformation Matrix and RR^T (3pts)

```
# Given position and velocity vectors
r = np.array([ephemeris_data['X'][0], ephemeris_data['Y'][0], ephemeris_data['Z'][0]])  # position vector in km
v = np.array([ephemeris_data['Vx'][0], ephemeris_data['Vy'][0], ephemeris_data['Vz'][0]])  # velocity vector in km/s

# Calculate the unit vectors for the RSW frame
R_hat = r / np.linalg.norm(r)  # Radial unit vector

W_hat = np.cross(r, v)  # Cross product
W_hat /= np.linalg.norm(W_hat)  # Cross-track unit vector

S_hat = np.cross(W_hat, R_hat)  # Along-track unit vector

# Construct the transformation matrix R
R = np.vstack((R_hat, S_hat, W_hat)).T

# Calculate R * R_transpose
RR_T = np.dot(R, R.T)

print("\nTransformation matrix R:")
print(R)
print("\nR * R^T (should be the identity matrix):")
print(RR_T)
```

```
Transformation matrix R:
[[-0.89210226 -0.37281864 -0.25526423]
 [ 0.05549682 -0.65108567  0.75697262]
 [-0.44841239  0.66113063  0.60152525]]

R * R^T (should be the identity matrix):
[[ 1.00000000e+00  2.77555756e-17  8.32667268e-17]
 [ 2.77555756e-17  1.00000000e+00 -2.22044605e-16]
 [ 8.32667268e-17 -2.22044605e-16  1.00000000e+00]]
```

Question 6

Find the relative error Δr between the ephemeris, and SGP4 position, for each TLE

```
# Calculate the difference vector
ephemeris_data['PositionVector'] = ephemeris_data.apply(lambda row: np.array([row['X'],
row['Y'], row['Z']]), axis=1)

ephemeris_data['differenceVectorTle1'] = ephemeris_data['tle1_position'] -
ephemeris_data['PositionVector']
ephemeris_data['differenceVectorTle2'] = ephemeris_data['tle2_position'] -
ephemeris_data['PositionVector']
ephemeris_data['differenceVectorTle3'] = ephemeris_data['tle3_position'] -
ephemeris_data['PositionVector']
```

```
# Apply the transformation matrix to each position vector
def apply_transformation(differenceVector, R):
    transformed_vector = np.dot(R, differenceVector)
    return transformed_vector
```

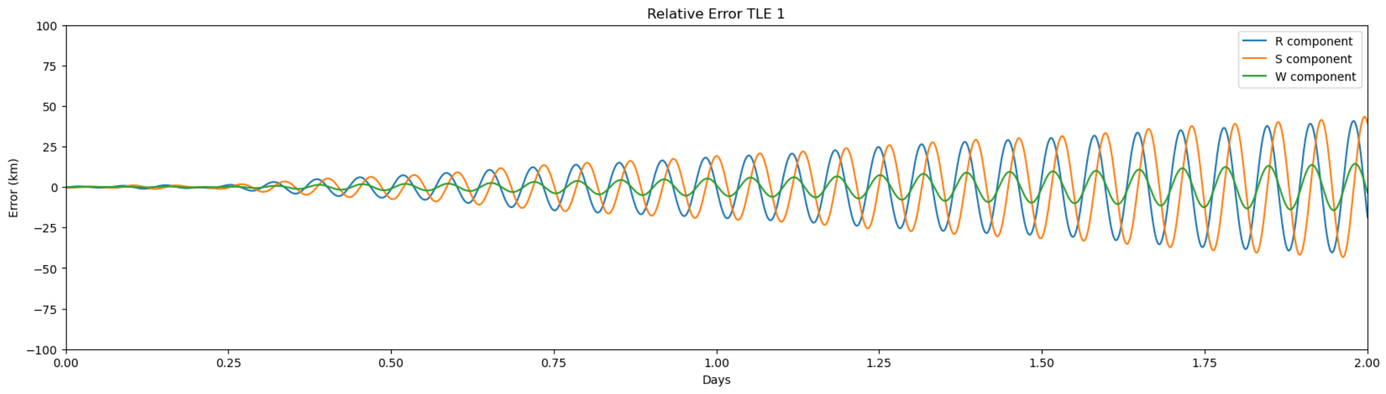
```
ephemeris_data[['X1_transformed', 'Y1_transformed', 'Z1_transformed']] =
ephemeris_data.apply(
    lambda row: pd.Series(apply_transformation(row['differenceVectorTle1'], R)), axis=1)
```

```
ephemeris_data[['X2_transformed', 'Y2_transformed', 'Z2_transformed']] =
ephemeris_data.apply(
    lambda row: pd.Series(apply_transformation(row['differenceVectorTle2'], R)), axis=1)
```

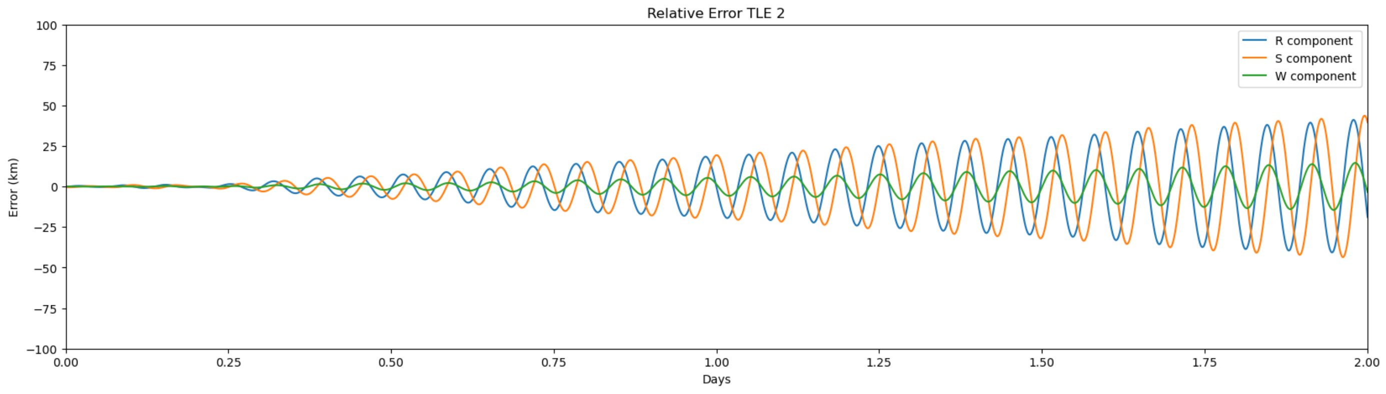
```
ephemeris_data[['X3_transformed', 'Y3_transformed', 'Z3_transformed']] =
ephemeris_data.apply(
    lambda row: pd.Series(apply_transformation(row['differenceVectorTle3'], R)), axis=1)
```

Plot the transformed vectors ($\hat{R}$, $\hat{S}$, $\hat{W}$) (15pts)

```
plt.figure(figsize=(20, 5))

# Plot X component
plt.plot(days, ephemeris_data['X1_transformed'], label='R component')

# Plot Y component
plt.plot(days, ephemeris_data['Y1_transformed'], label='S component')

# Plot Z component
plt.plot(days, ephemeris_data['Z1_transformed'], label='W component')

plt.xlabel('Days')
plt.ylabel('Error (km)')
plt.xlim(0, 2)
plt.ylim(-100, 100)
plt.title('Relative Error TLE 1')
plt.legend()
plt.show()
```

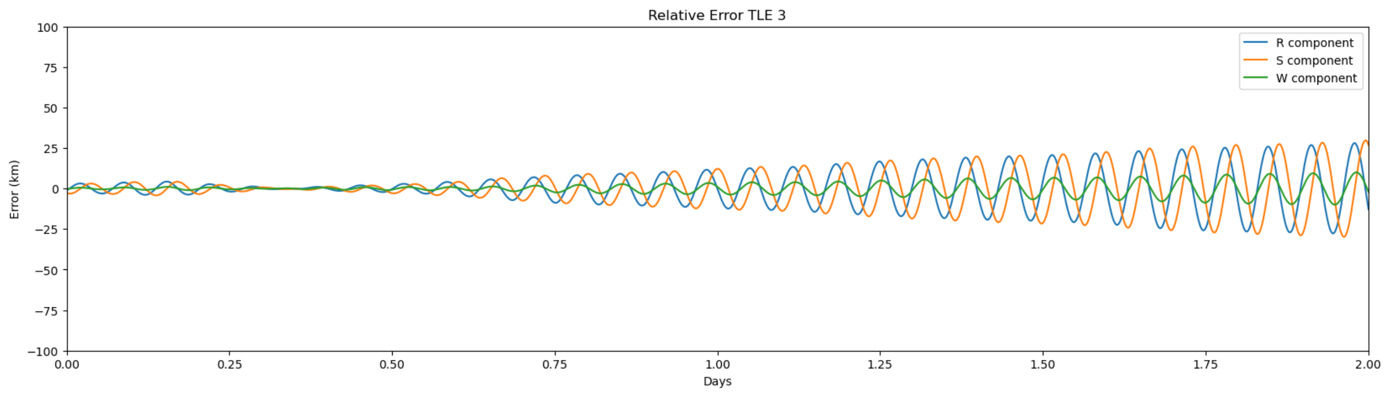


```python
plt.figure(figsize=(20, 5))
# Plot X component
plt.plot(days, ephemeris_data['X2_transformed'], label='R component')

# Plot Y component
plt.plot(days, ephemeris_data['Y2_transformed'], label='S component')

# Plot Z component
plt.plot(days, ephemeris_data['Z2_transformed'], label='W component')

plt.xlim(0, 2)
plt.ylim(-100, 100)
plt.xlabel('Days')
plt.ylabel('Error (km)')
plt.title('Relative Error TLE 2')
plt.legend()
plt.show()
```



```python
plt.figure(figsize=(20, 5))

# Plot X component
plt.plot(days, ephemeris_data['X3_transformed'], label='R component')

# Plot Y component
plt.plot(days, ephemeris_data['Y3_transformed'],label='S component')

# Plot Z component
```

```
plt.plot(days, ephemeris_data['Z3_transformed'], label='W component')

plt.xlim(0, 2)
plt.ylim(-100, 100)
plt.xlabel('Days')
plt.ylabel('Error (km)')
plt.title('Relative Error TLE 3')
plt.legend()
plt.show()
```



Axis with the greatest error magnitude (3pts)

The S axis has the highest error for all the graphs. This is the along track component.

Estimate of error growth per day (5pts)

I would approximate the error by eye at around **20 km per day**. But the rate does not seem consistent for all ranges and gets worse for the older TLEs.

In predicting of 3 days for the near-earth objects, the positional errors are less than
40 km [1].

Given this the positional error can be roughly calculated as 40/3 = 13.3 km/day

[1]W. Dong and Z. Chang-Yin, "11-18; revised version 2008-12-01 A translation of," Chinese Astronomy and Astrophysics, vol. 34, no. 3, pp. 332–339, 2010, doi: https://doi.org/10.1016/j.chinastron.2009.12.009.

Potential causes of significant variation from True Orbit (3pts)

- Outdated TLEs
- Inaccuracy in satellite tracking
- Significant pertubation effects not accounted for by SGP4

Question 7

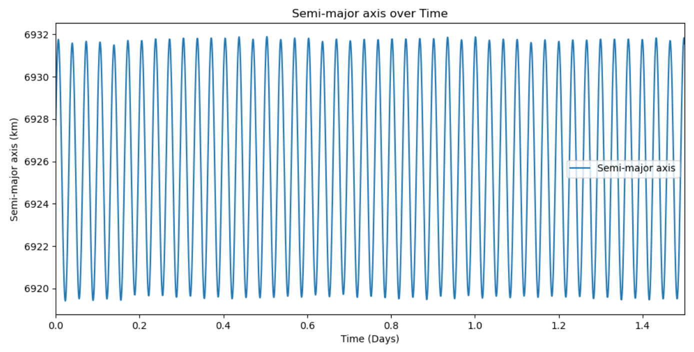
Find orbital elements

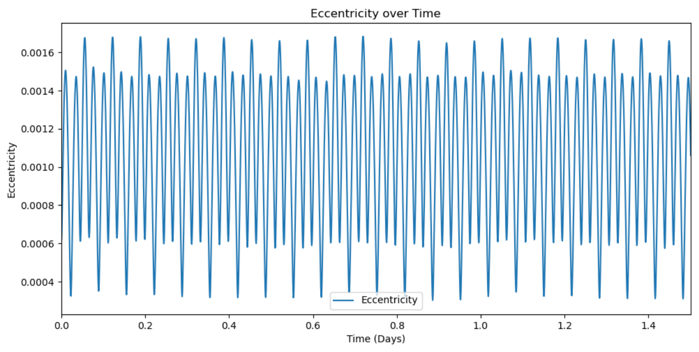
```python
from poliastro.bodies import Earth
from poliastro.twobody import Orbit
from poliastro.core.angles import E_to_M, nu_to_E

# Empty lists to hold the calculated orbital elements
a_list = []
ecc_list = []
inc_list = []
raan_list = []
argp_list = []
nu_list = []
mean_anomaly_list = []

for index, row in ephemeris_data.iterrows():
    # Create a position vector›
    r = [row['X'], row['Y'], row['Z']] * u.km
    # Create a velocity vector
    v = [row['Vx'], row['Vy'], row['Vz']] * (1000 * u.m / u.s)

    # Create an orbit from the vectors
    orbit = Orbit.from_vectors(Earth, r, v)

    # Extract the orbital elements
    a, ecc, inc, raan, argp, nu = orbit.classical()

    # Calculate the Mean Anomaly
    E = nu_to_E(nu.to(u.rad).value, ecc.value) * u.rad
    M = E_to_M(E.value, ecc.value) * u.rad

    # Append the results to lists
    a_list.append(a.to(u.km).value)
    ecc_list.append(ecc.value)
    inc_list.append(inc.to(u.deg).value)
    raan_list.append(raan.to(u.deg).value)
    argp_list.append(argp.to(u.deg).value)
    nu_list.append((nu.to(u.deg).value) % 360)
    mean_anomaly_list.append((M.to(u.deg).value) % 360)

# Add the calculated elements to the DataFrame
ephemeris_data['semi_major_axis'] = a_list
ephemeris_data['eccentricity'] = ecc_list
ephemeris_data['inclination'] = inc_list
ephemeris_data['raan'] = raan_list
ephemeris_data['arg_perigee'] = argp_list
ephemeris_data['true_anomaly'] = nu_list
ephemeris_data['mean_anomaly'] = mean_anomaly_list
```

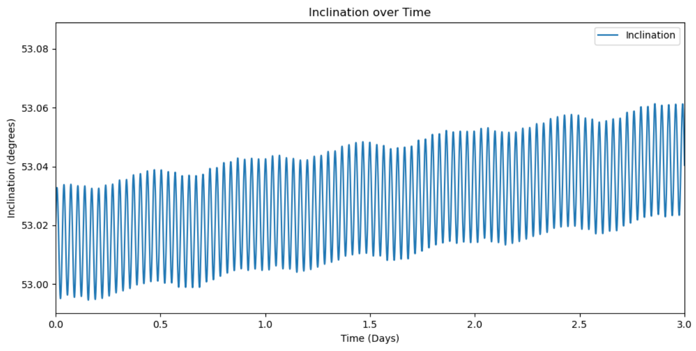
```python
# Calculate the differences between subsequent RAAN values
ephemeris_data['RAAN_diff'] = ephemeris_data['raan'].diff()

# Calculate the derivative (rate of change of RAAN)
ephemeris_data['RAAN_derivative'] = ephemeris_data['RAAN_diff'] / 0.00069 #Days per entry
```

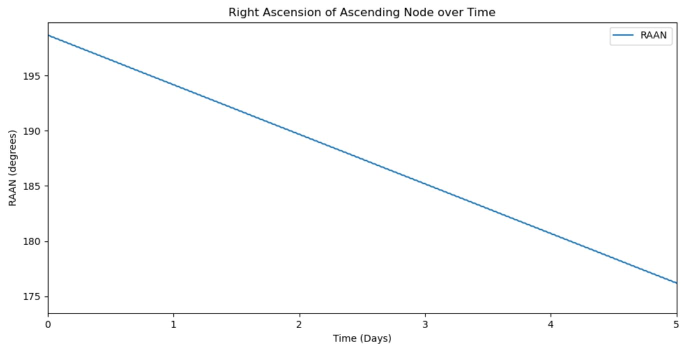
Plot Orbital Elements (32pts)

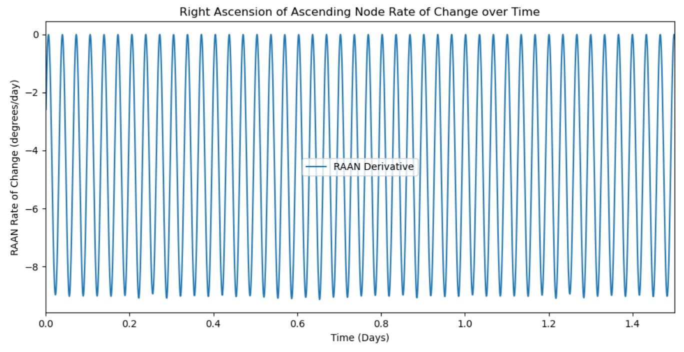
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 20))

# Plot Semi-major axis
plt.subplot(4, 2, 1)
plt.plot(days, ephemeris_data['semi_major_axis'], label='Semi-major axis')
plt.xlabel('Time (Days)')
plt.ylabel('Semi-major axis (km)')
plt.title('Semi-major axis over Time')
plt.xlim(0, 1.5)
plt.legend()

# Plot Eccentricity
plt.subplot(4, 2, 2)
plt.plot(days, ephemeris_data['eccentricity'], label='Eccentricity')
plt.xlabel('Time (Days)')
plt.ylabel('Eccentricity')
plt.title('Eccentricity over Time')
plt.xlim(0, 1.5)
plt.legend()

# Plot Inclination
plt.subplot(4, 2, 3)
plt.plot(days, ephemeris_data['inclination'], label='Inclination')
plt.xlabel('Time (Days)')
plt.ylabel('Inclination (degrees)')
plt.title('Inclination over Time')
plt.xlim(0, 3)
plt.legend()

# Plot RAAN
plt.subplot(4, 2, 4)
plt.plot(days, ephemeris_data['raan'], label='RAAN')
plt.xlabel('Time (Days)')
plt.ylabel('RAAN (degrees)')
plt.title('Right Ascension of Ascending Node over Time')
plt.xlim(0, 5)
plt.legend()

# Plot RAAN Derivative
plt.subplot(4, 2, 5)
plt.plot(days, ephemeris_data['RAAN_derivative'], label='RAAN Derivative')
plt.xlabel('Time (Days)')
plt.ylabel('RAAN Rate of Change (degrees/day)')
plt.title('Right Ascension of Ascending Node Rate of Change over Time')
plt.xlim(0, 1.5)
plt.legend()

# Plot Argument of Perigee
plt.subplot(4, 2, 6)
plt.plot(days, ephemeris_data['arg_perigee'], label='Argument of Perigee')
plt.xlabel('Time (Days)')
plt.ylabel('Argument of Perigee (degrees)')
plt.title('Argument of Perigee over Time')
plt.xlim(0, 1.5)
plt.legend()

# Plot True Anomaly
plt.subplot(4, 2, 7)
plt.plot(days, ephemeris_data['true_anomaly'], label='True Anomaly')
plt.xlabel('Time (Days)')
plt.ylabel('True Anomaly (Deg)')
plt.title('True Anomaly over Time')
plt.xlim(0, 1.5)
plt.legend()

# Plot Mean Anomaly
plt.subplot(4, 2, 8)
plt.plot(days, ephemeris_data['mean_anomaly'], label='Mean Anomaly')
plt.xlabel('Time (Days)')
plt.ylabel('Mean Anomaly (Deg)')
plt.title('Mean Anomaly over Time')
plt.xlim(0, 1.5)
plt.legend()

plt.tight_layout()
plt.show()
```

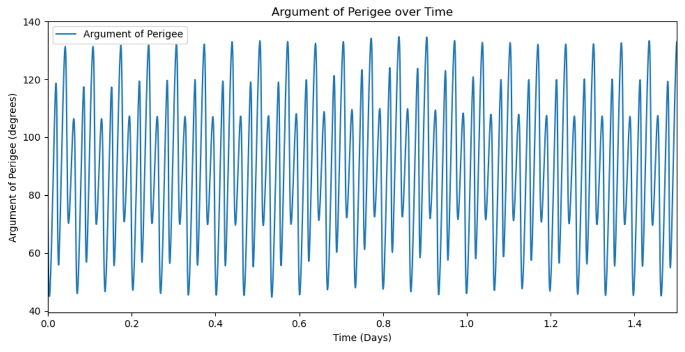


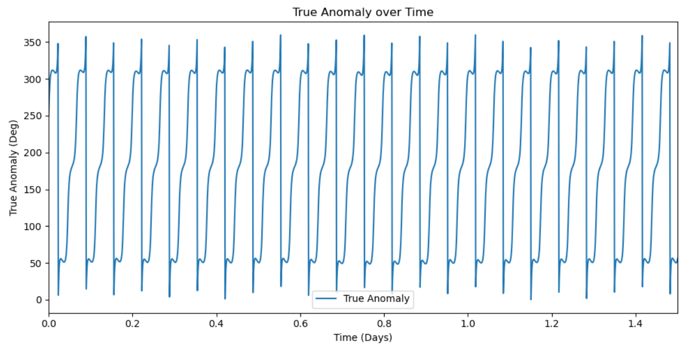


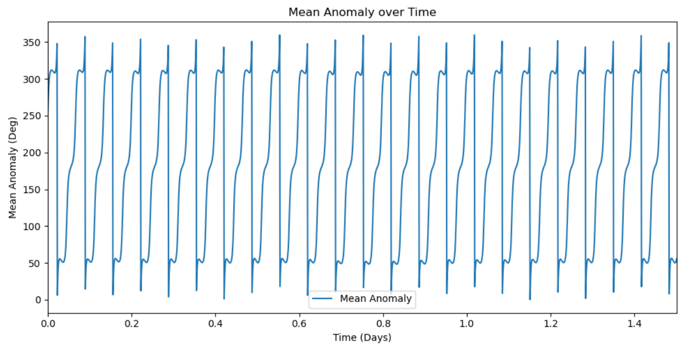












Range of the X axis has been reduced for readability and feature resolution. Most are over 1.5 Days. RAAN is over the full 5 but this does reduce the resolution. If you zoom in you can see there are cyclical variations over time.

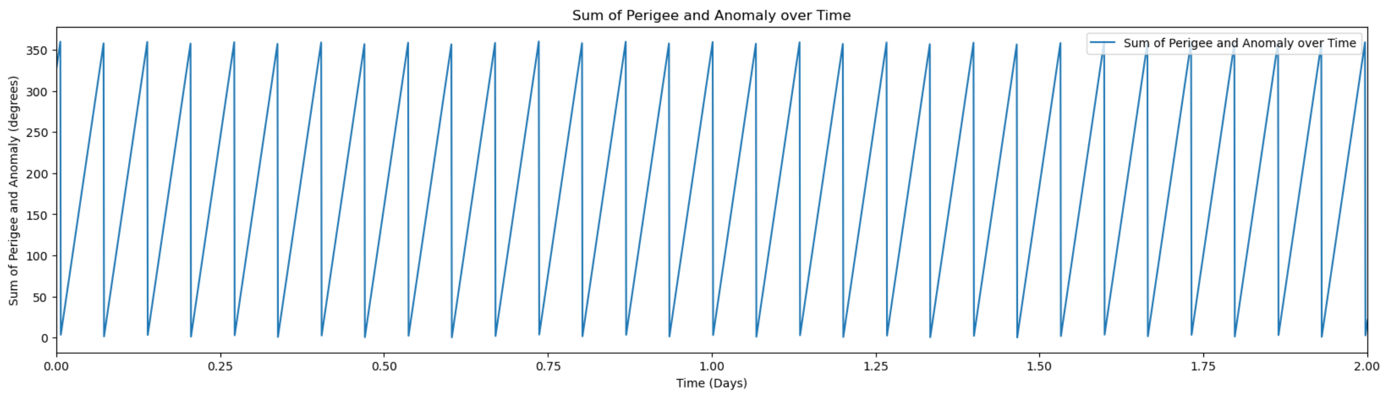
Complexity in Mean and True Anomaly (7pts)

This is likely due to the perigee fluctuating and changing the point from which the mean and true anomaly are measured. This fluctuation occurs due to the low eccentricity of the orbit so the perigee is not a clear point on the orbit and can move. If the orbit was more eccentric the perigee would be alot more regular.

Sum together the true anomaly and argument of peregee (5pts)

```
ephemeris_data['anomalyPergeeSum'] = ephemeris_data.apply(lambda row: (row['arg_perigee']
+ row['true_anomaly']) % 360, axis=1)
```

```
# Plot Sum of perigee and anomaly
plt.figure(figsize=(20, 5))
plt.plot(days, ephemeris_data['anomalyPergeeSum'], label='Sum of Perigee and Anomaly over
Time')
plt.xlabel('Time (Days)')
plt.ylabel('Sum of Perigee and Anomaly (degrees)')
plt.title('Sum of Perigee and Anomaly over Time')
plt.xlim(0, 2)
plt.legend()
plt.show()
```



Why is it so perfect?

The sum of Angle of Perigee and True Anomaly gives the Argument of Latitude. This is useful for orbits with low eccentricity because the perigee is not well defined. Using the argument of latitude ensures the measurement is more consitent and disconnected from variations of the perigee.

Question 8

Calculate average rate of change of the right ascension of the ascending node (2pts)

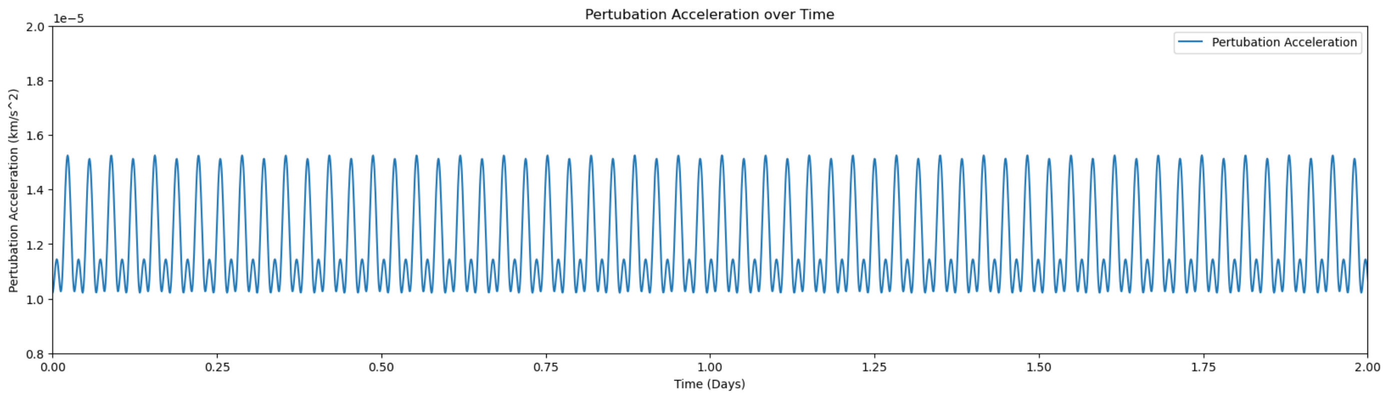
```
RAAN_derivative_Average = ephemeris_data['RAAN_derivative'].mean()
print(f"Nodal regression rate average (degrees/day): {RAAN_derivative_Average}")
```

```
Nodal regression rate average (degrees/day): -4.518600440632246
```

Calculate and plot pertubation acceleration (5pts)

```python
def j2_acceleration(x, y, z, mu_earth, r_earth, j2):
    r = np.sqrt(x**2 + y**2 + z**2)
    factor = -3/2*j2*mu_earth*(r_earth/r)**2/r**2
    x_factor = x/r*(1 - 5*z**2/r**2)
    y_factor = y/r*(1 - 5*z**2/r**2)
    z_factor = z/r*(3 - 5*z**2/r**2)
    return np.linalg.norm(factor * np.array([x_factor, y_factor, z_factor]))


# Constants
mu_earth = 398600.4418 # in km^3/s^2
r_earth = 6378.137 # in km
j2 = 1.08262668e-3 # J2 oblateness factor

# Calculate the J2 perturbation acceleration
ephemeris_data['j2Acceleration'] = ephemeris_data.apply(lambda row: j2_acceleration(
    row['X'],
    row['Y'],
    row['Z'],
    mu_earth, r_earth, j2), axis=1)
```

```python
# Plot Pertubation Acceleration over Time
plt.figure(figsize=(20, 5))
plt.plot(days, ephemeris_data['j2Acceleration'], label='Pertubation Acceleration')
plt.xlabel('Time (Days)')
plt.ylabel('Pertubation Acceleration (km/s^2)')
plt.title('Pertubation Acceleration over Time')
plt.xlim(0, 2)
plt.ylim(0.000008, 0.00002)
plt.legend()
plt.show()
```



Theoretical Value for Rate of Change of RAAN (5pts)

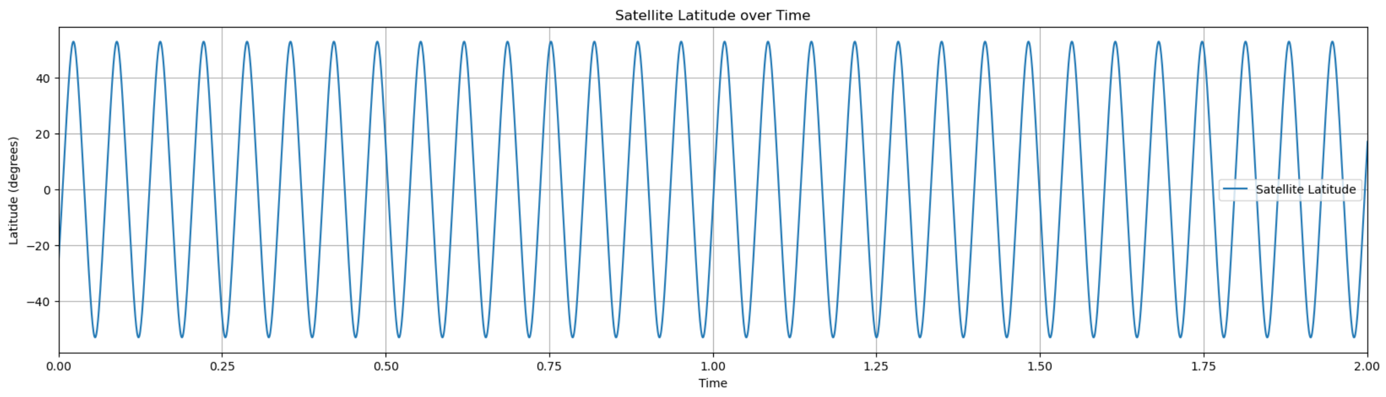
```
# Constants
G = 6.67430e-11  # Gravitational constant in m^3 kg^-1 s^-2
M_earth = 5.972e24  # Mass of Earth in kg
R_earth = 6378.137e3  # Radius of Earth in meters

# Parameters
j2 = 1.08262668e-3 # J2 oblateness factor
a = ephemeris_data['semi_major_axis'].mean() * 1000  # Semi-major axis in meters
e = ephemeris_data['eccentricity'].mean()  # Eccentricity of the orbit
i_degrees = ephemeris_data['inclination'].mean()  # Inclination in degrees

# Convert inclination to radians
i = np.radians(i_degrees)

# Calculate semi-latus rectum
l = a * (1 - e**2)

# Mean motion
m = np.sqrt(G * M_earth / a**3)  # In rad/s

# Calculate nodal regression rate
omega_dot = (-3/2) * j2 * (R_earth / l)**2 * m * np.cos(i)

# Convert rad/s to degrees/day
omega_dot_degrees_day = np.degrees(omega_dot) * 60 * 60 * 24

print(f"Nodal regression rate (degrees/day): {omega_dot_degrees_day}")
```

```
Nodal regression rate (degrees/day): -4.490718291986784
```

Plot latitude over time (5pts)

```
# Calculate latitude from the x, y, z coordinates
ephemeris_data['Latitude'] = np.degrees(np.arctan2(ephemeris_data['Z'],
np.sqrt(ephemeris_data['X']**2 + ephemeris_data['Y']**2)))
```

```
# Plot the latitude over time
plt.figure(figsize=(20, 5))
plt.plot(days, ephemeris_data['Latitude'], label='Satellite Latitude')
plt.xlabel('Time')
plt.ylabel('Latitude (degrees)')
plt.title('Satellite Latitude over Time')
plt.legend()
plt.grid(True)
plt.xlim(0, 2)
plt.show()
```



Consideration of the compared graphs (Latitide and Pertubation Acceleration)

The peaks of maximum and minimum latitude match the spikes in pertubation acceleration. This would imply a larger pertubation effect as the satellite moves away from the equator on either the north or south side. This could be due to the bulge around the equator creating a tug when the satellite is at other latitudes. So the magnitude of the force changes as the satellite moves away from the equator and back again.