GNSS Homework 3

Khushaldas Badhan

# Transformation between Keplerian parameters and Cartesian parameters 𝑟 and 𝑟̇ Write a program for doing the transformation between the state vector containing the Keplerian elements {𝑎,𝑒,𝐼, 𝛺, 𝜔, 𝑀} and the Cartesian state vector {𝑟,𝑟̇}. To validate your code, you should be able to do the transformation back and forth and check if you can get the same inputs and outputs.

Below is the code for transformation between the state vector containing the Keplerian elements {𝑎,𝑒,𝐼, 𝛺, 𝜔, 𝑀} and the Cartesian state vector {𝑟,𝑟̇}.

1. import numpy as np

2. from scipy.optimize import newton

3.

4. def mean\_to\_eccentric\_anomaly(M, e, tol=1e-10):

5. """Solve for Eccentric Anomaly (E) given Mean Anomaly (M) and eccentricity (e)"""

6. func = lambda E: E - e \* np.sin(E) - M

7. E0 = M if e < 0.8 else np.pi

8. return newton(func, E0, tol=tol)

9.

10. def keplerian\_to\_cartesian(a, e, I, Omega, omega, M, mu):

11. """Convert from Keplerian elements to Cartesian state vector."""

12. # Convert to radians

13. I, Omega, omega = np.radians(I), np.radians(Omega), np.radians(omega)

14.

15. # Calculate the Eccentric Anomaly

16. E = mean\_to\_eccentric\_anomaly(M, e)

17.

18. # True Anomaly

19. nu = 2 \* np.arctan(np.sqrt((1 + e) / (1 - e)) \* np.tan(E / 2))

20.

21. # Perifocal coordinates

22. r\_peri = a \* (1 - e \* np.cos(E))

23. r\_vec\_peri = np.array([r\_peri \* np.cos(nu), r\_peri \* np.sin(nu), 0])

24.

25. # Adjusting velocity vector calculation

26. v\_vec\_peri = np.array([-np.sin(E), np.sqrt(1 - e\*\*2) \* np.cos(E), 0])

27. v\_vec\_peri \*= np.sqrt(mu \* a) / r\_peri / np.sqrt(1 - e\*\*2)

28.

29. # Rotation Matrices

30. R\_z\_Omega = np.array([[np.cos(Omega), -np.sin(Omega), 0], [np.sin(Omega), np.cos(Omega), 0], [0, 0, 1]])

31. R\_x\_I = np.array([[1, 0, 0], [0, np.cos(I), -np.sin(I)], [0, np.sin(I), np.cos(I)]])

32. R\_z\_omega = np.array([[np.cos(omega), -np.sin(omega), 0], [np.sin(omega), np.cos(omega), 0], [0, 0, 1]])

33. R = R\_z\_Omega @ R\_x\_I @ R\_z\_omega

34.

35. # Transform to Ecliptic Coordinates

36. r\_vec = R @ r\_vec\_peri

37. v\_vec = R @ v\_vec\_peri

38.

39. return r\_vec, v\_vec

40.

41. def cartesian\_to\_keplerian(r, v, mu):

42. # Specific angular momentum

43. h = np.cross(r, v)

44. n = np.cross([0, 0, 1], h)

45.

46. # Eccentricity vector

47. e\_vector = np.cross(v, h) / mu - r / np.linalg.norm(r)

48. e = np.linalg.norm(e\_vector)

49.

50. # Semi-major axis

51. energy = np.linalg.norm(v)\*\*2 / 2 - mu / np.linalg.norm(r)

52. a = -mu / (2 \* energy)

53.

54. # Inclination

55. I = np.degrees(np.arccos(h[2] / np.linalg.norm(h))) # Converted to degrees

56.

57. # RAAN

58. RAAN = np.degrees(np.arccos(n[0] / np.linalg.norm(n))) # Converted to degrees

59. if n[1] < 0:

60. RAAN = 360 - RAAN

61.

62. # Argument of periapsis

63. w = np.degrees(np.arccos(np.dot(n, e\_vector) / (np.linalg.norm(n) \* e))) # Converted to degrees

64. if e\_vector[2] < 0:

65. w = 360 - w

66.

67. # True anomaly

68. nu = np.degrees(np.arccos(np.dot(e\_vector, r) / (e \* np.linalg.norm(r)))) # Converted to degrees

69. if np.dot(r, v) < 0:

70. nu = 360 - nu

71.

72. # Mean Anomaly (M)

73. E = 2 \* np.arctan(np.tan(nu / 2) / np.sqrt((1 + e) / (1 - e)))

74. M = np.degrees(E - e \* np.sin(E)) # Converted to degrees

75.

76. # Normalize M to be within 0 to 360 degrees

77. M = M % 360

78.

79.

80. return a, e, I, RAAN, w, M

81.

82. if \_\_name\_\_ == "\_\_main\_\_":

83. # Example usage

84. mu = 398600 # Gravitational parameter for Earth [km^3/s^2]

85. a, e, I, Omega, omega, M = 7000, 0.001, 30, 40, 50, 0 # Keplerian elements

86.

87. r\_vec, v\_vec = keplerian\_to\_cartesian(a, e, I, Omega, omega, M, mu)

88. print("Cartesian State Vector:")

89. print("Position Vector (r):", r\_vec)

90. print("Velocity Vector (v):", v\_vec)

91.

92. # Convert back from Cartesian to Keplerian

93. a\_back, e\_back, I\_back, Omega\_back, omega\_back, M\_back = cartesian\_to\_keplerian(r\_vec, v\_vec, mu)

94. print("\nConverted Back to Keplerian Elements:")

95. print("Semi-major axis (a):", a\_back)

96. print("Eccentricity (e):", e\_back)

97. print("Inclination (I):", I\_back)

98. print("RAAN (Omega):", Omega\_back)

99. print("Argument of Periapsis (omega):", omega\_back)

100. print("Mean Anomaly (M):", M\_back)

101.

# Visualize the 2nd law of Kepler (area law)

In the zip file HW3Ex2.

The orbit data is calculated using the k2cloop.py python script and saved in orbit\_data.csv file. This data is then used in the matlab script plot\_orbit.m for plotting orbit.

# Write down the equations to get position and velocity from orbital elements

First we need to calculate eccentric anomaly. This is calculated by Kepler's equation using an iterative process

Then we need to calculate the true anomaly

Next we can find r, i.e. distance of the orbiting body to central body.

Then we can calculate the positions in the orbital plane, X’ and Y’

Next we can calculate the velocities in the orbital plane and

To transform these positions and velocities in orbital plane into the inertial frame, we can apply series of rotations.

Here,

Here, is rotation angle.

# Orbital elements from position and velocity

For this we first need to calculate Specific Angular Momentum

Then we can calculate inclination

Then we can calculate the node line vector

Next Right Ascension of the Ascending Node

If

Eccentricity Vector

Eccentricity

Argument of Periapsis

If

True Anomaly

If

Mean Anomaly

Semi-major axis

# Compute the Keplerian elements for an Earth-orbiting satellite

Computed Keplerian elements:  
Keplerian Elements:

Semi-major axis (a): 25015.18669097937 km

Eccentricity (e): 0.7079768603248031

Inclination (I): 6.970729208730736 degrees

Right Ascension of Ascending Node (RAAN): 173.290163192243 degrees

Argument of Periapsis (w): 91.55290111000984 degrees

Mean Anomaly (M): 15.448295945846853 degrees

For the calculations below Python script was used. This script uses cartesian\_to\_keplerian function from the script in **1**.

1. import numpy as np

2. from Orbit\_conversion import cartesian\_to\_keplerian

3.

4. mu = 398600 # Gravitational parameter for Earth [km^3/s^2]

5.

6. # Define the position and velocity vectors in km and km/s, respectively

7. r = np.array([10000.0, 40000.0, -5000.0]) # Position vector in km

8. v = np.array([-1.5, 1.0, -0.1]) # Velocity vector in km/s

9.

10. # Call the function

11. keplerian\_elements = cartesian\_to\_keplerian(r, v, mu)

12.

13. # Print the results

14. print("Keplerian Elements:")

15. print(f"Semi-major axis (a): {keplerian\_elements[0]} km")

16. print(f"Eccentricity (e): {keplerian\_elements[1]}")

17. print(f"Inclination (I): {keplerian\_elements[2]} degrees")

18. print(f"Right Ascension of Ascending Node (RAAN): {keplerian\_elements[3]} degrees")

19. print(f"Argument of Periapsis (w): {keplerian\_elements[4]} degrees")

20. print(f"Mean Anomaly (M): {keplerian\_elements[5]} degrees")

# Orbit prediction

Below is the code for predicting the Keplerian orbit. The file predicted\_orbit.csv is the cartesian position coordinates of the satellite.

1. import numpy as np

2. from scipy.optimize import newton

3. import csv

4.

5. # Constants

6. G = 6.67430e-11 # Gravitational constant in m^3 kg^-1 s^-2

7. M\_earth = 5.972e24 # Mass of Earth in kg

8. mu = G \* M\_earth # Gravitational parameter for Earth in m^3/s^2

9.

10. # Provided functions

11. def mean\_to\_eccentric\_anomaly(M, e, tol=1e-10):

12. """Solve for Eccentric Anomaly (E) given Mean Anomaly (M) and eccentricity (e)"""

13. func = lambda E: E - e \* np.sin(E) - M

14. E0 = M if e < 0.8 else np.pi

15. return newton(func, E0, tol=tol)

16.

17. def keplerian\_to\_cartesian(a\_km, e, I, Omega, omega, M, mu):

18. """Convert from Keplerian elements to Cartesian state vector."""

19. # Convert semi-major axis to meters for calculation

20. a = a\_km \* 1000 # Convert km to m

21.

22. I, Omega, omega = np.radians(I), np.radians(Omega), np.radians(omega)

23.

24. E = mean\_to\_eccentric\_anomaly(M, e)

25.

26. nu = 2 \* np.arctan(np.sqrt((1 + e) / (1 - e)) \* np.tan(E / 2))

27.

28. r\_peri = a \* (1 - e \* np.cos(E))

29. r\_vec\_peri = np.array([r\_peri \* np.cos(nu), r\_peri \* np.sin(nu), 0])

30.

31. v\_vec\_peri = np.array([-np.sin(E), np.sqrt(1 - e\*\*2) \* np.cos(E), 0])

32. v\_vec\_peri \*= np.sqrt(mu \* a) / r\_peri / np.sqrt(1 - e\*\*2)

33.

34. R\_z\_Omega = np.array([[np.cos(Omega), -np.sin(Omega), 0], [np.sin(Omega), np.cos(Omega), 0], [0, 0, 1]])

35. R\_x\_I = np.array([[1, 0, 0], [0, np.cos(I), -np.sin(I)], [0, np.sin(I), np.cos(I)]])

36. R\_z\_omega = np.array([[np.cos(omega), -np.sin(omega), 0], [np.sin(omega), np.cos(omega), 0], [0, 0, 1]])

37. R = R\_z\_Omega @ R\_x\_I @ R\_z\_omega

38.

39. r\_vec = R @ r\_vec\_peri

40. v\_vec = R @ v\_vec\_peri

41.

42. # Convert results to km and km/s for output

43. return r\_vec / 1000, v\_vec / 1000

44.

45. # Function to calculate the orbital period

46. def orbital\_period(a\_km):

47. a = a\_km \* 1000 # Convert km to m for calculation

48. return 2 \* np.pi \* np.sqrt(a\*\*3 / mu)

49.

50. # Function to predict the orbit for 16 revolutions

51. def predict\_orbit\_and\_write\_csv(a\_km, e, I, Omega, omega, nu, filename="satellite\_orbit.csv", num\_points\_per\_orbit=100):

52. T = orbital\_period(a\_km) # Orbital period

53. delta\_t = T / num\_points\_per\_orbit # Time step

54.

55. orbit\_positions = []

56.

57. with open(filename, 'w', newline='') as file:

58. writer = csv.writer(file)

59. writer.writerow(["X (km)", "Y (km)", "Z (km)", "Time Step", "Revolution"])

60.

61. for revolution in range(16):

62. for step in range(num\_points\_per\_orbit):

63. M = 2 \* np.pi \* step \* delta\_t / T # Mean anomaly

64. r\_vec, \_ = keplerian\_to\_cartesian(a\_km, e, I, Omega, omega, M, mu)

65. orbit\_positions.append(r\_vec)

66.

67. # Write to CSV: position (in km) and time information

68. writer.writerow([r\_vec[0], r\_vec[1], r\_vec[2], step, revolution])

69.

70. return np.array(orbit\_positions)

71.

72.

73.

74. if \_\_name\_\_ == "\_\_main\_\_":

75. # Example orbital elements (replace with actual values)

76. a = 6827.9986 # Semi-major axis in kilometers (including Earth's radius if altitude is given)

77. e = 0.004 # Eccentricity

78. I = 87.3 # Inclination in degrees

79. Omega = 0 # Right Ascension of Ascending Node in degrees

80. omega = 0 # Argument of Perigee in degrees

81. nu = 0.2106 # True Anomaly in degrees

82.

83. # Predict the orbit and write to CSV

84. orbit\_positions = predict\_orbit\_and\_write\_csv(a, e, I, Omega, omega, nu, r"C:\Users\KhushaldasBadhan\OneDrive - ODYSSEUS SPACE\Documents\semester\GNSS\HW3\predicted\_orbit.csv")

85.

86. # Show the shape of the orbit positions array

87. orbit\_positions.shape # This should give us 16 orbits with 100 points each

88.

89. import matplotlib.pyplot as plt

90. from mpl\_toolkits.mplot3d import Axes3D

91.

92. # Function to plot the orbit

93. def plot\_orbit(orbit\_positions):

94. fig = plt.figure(figsize=(10, 8))

95. ax = fig.add\_subplot(111, projection='3d')

96.

97. # Plot the orbit

98. ax.plot(orbit\_positions[:, 0], orbit\_positions[:, 1], orbit\_positions[:, 2], label='Satellite Orbit')

99.

100. # Plot Earth (assuming a perfect sphere for simplicity)

101. u, v = np.mgrid[0:2\*np.pi:20j, 0:np.pi:10j]

102. x = Earth\_radius \* np.cos(u) \* np.sin(v)

103. y = Earth\_radius \* np.sin(u) \* np.sin(v)

104. z = Earth\_radius \* np.cos(v)

105. ax.plot\_surface(x, y, z, color='b', alpha=0.3)

106.

107. # Axes labels

108. ax.set\_xlabel('X (m)')

109. ax.set\_ylabel('Y (m)')

110. ax.set\_zlabel('Z (m)')

111. ax.set\_title('3D Orbit Plot')

112.

113. plt.legend()

114. plt.show()

115.

116. # Radius of the Earth

117. Earth\_radius = 6371e3 # in meters

118.

119. # Plot the orbit

120. plot\_orbit(orbit\_positions)