ENAE 404 - 0101

Homework 02: 2BP

Due on February 25th, 2025 at 09:30 AM

Dr. Barbee, 09:30

Vai Srivastava

February 24, 2025

Problem 1:

Given the following position and velocity vectors, calculate the Keplerian orbital elements, assuming Earth is the central body. Do not use a computer code to do this. Vectors are in units of km and $\frac{\text{km}}{\text{s}}$.

$$\vec{r} = 3634.1\hat{\boldsymbol{x}} + 5926\hat{\boldsymbol{y}} + 1206.6\hat{\boldsymbol{z}}$$
$$\vec{v} = -6.9049\hat{\boldsymbol{x}} + 4.3136\hat{\boldsymbol{y}} + 2.6163\hat{\boldsymbol{z}}$$

Solution

$$\begin{aligned}
\mu_\oplus &= 398\,600\,\frac{\text{km}^3}{\text{s}^2} \\
r = \|\mathbf{r}\| &= \sqrt{3634.1^2 + 5926^2 + 1206.6^2} \approx 7055 \text{ km} \\
v = \|\mathbf{v}\| &= \sqrt{(-6.9049)^2 + 4.3136^2 + 2.6163^2} \approx 8.55 \text{ km/s} \\
\mathbf{h} &= \mathbf{r} \times \mathbf{v} \\
h = \|\mathbf{h}\| &\approx 6.02 \times 10^4 \text{ km}^2/\text{s} \\
\mathcal{E} = \frac{v^2}{2} - \frac{\mu_\oplus}{r} \approx \frac{73.16}{2} - \frac{398600}{7055} &\approx 36.58 - 56.50 \approx -19.92 \text{ km}^2/\text{s}^2 \\
a &= -\frac{\mu_\oplus}{2\mathcal{E}} \\
a &\approx \frac{398600}{39.84} \approx 1 \times 10^4\,\text{km} \quad \square \\
e &= \sqrt{1 + \frac{2\mathcal{E}\,h^2}{\mu_\oplus^2}} \approx 0.30 \quad \square \\
i &= \arccos\left(\frac{h_z}{h}\right) \approx \arccos(0.939) \approx 20^\circ \quad \square \\
\mathbf{n} &= \hat{\mathbf{k}} \times \mathbf{h} \\
\|\mathbf{n}\| &\approx 2.06 \times 10^4\,\frac{\text{km}^2}{\text{s}} \quad \square \\
\Omega &= \arccos\left(\frac{n_x}{\|\mathbf{n}\|}\right) \approx \arccos(0.8660) \approx 30^\circ \quad \square \\
\omega &= \arccos\left(\frac{\mathbf{n}\cdot\mathbf{e}}{\|\mathbf{n}\|\,e}\right) \\
\omega &\approx 15^\circ \quad \square \\
\nu &= \arccos\left(\frac{\mathbf{e}\cdot\mathbf{r}}{e\,r}\right) \\
\nu &\approx 30^\circ \quad \square
\end{aligned}$$

$$\boxed{\begin{aligned}
a &\approx 1 \times 10^4\,\text{km} \\
e &\approx 0.30 \\
i &\approx 20^\circ \\
\Omega &\approx 30^\circ \\
\omega &\approx 15^\circ \\
\nu &\approx 15^\circ
\end{aligned}}$$

Problem 2:

1. Write code to convert form Cartesian coordinates to orbital elements.
2. Using subplot, plot the osculating orbital elements for the orbit of Didymos from HW00.
3. Describe why your plots make sense (in reference to both the time variation of the orbital elements as well as the plot of the orbit in 3D space).

Solution

Part A

```python
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

mu_sun = 1.32712440018e11


def state_to_keplerian(r_vec, v_vec, mu):
    """
    Convert Cartesian state vectors to orbital elements.

    Parameters:
      r_vec: Position vector (km)
      v_vec: Velocity vector (km/s)
      mu:    Gravitational parameter (km^3/s^2)

    Returns:
      a          : semimajor axis (km)
      e          : eccentricity (unitless)
      inc        : inclination (rad)
      RAAN       : right ascension of the ascending node (rad)
      arg_peri   : argument of perigee (rad)
      nu         : true anomaly (rad)
    """
    r = np.array(r_vec)
    v = np.array(v_vec)
    r_norm = np.linalg.norm(r)
    v_norm = np.linalg.norm(v)

    # Specific angular momentum vector and magnitude
    h = np.cross(r, v)
    h_norm = np.linalg.norm(h)

    # Inclination
    inc = np.arccos(h[2] / h_norm)

    # Node vector (pointing toward ascending node)
    K = np.array([0, 0, 1])
    n = np.cross(K, h)
    n_norm = np.linalg.norm(n)

    # Eccentricity vector and eccentricity magnitude
    e_vec = (np.cross(v, h) / mu) - (r / r_norm)
    e = np.linalg.norm(e_vec)

    # Semimajor axis (using vis-viva equation)
    a = 1 / (2 / r_norm - v_norm**2 / mu)
```

Problem 2 continued on next page. . .

```python

    # RAAN
    if n_norm > 1e-8:
        RAAN = np.arccos(n[0] / n_norm)
        if n[1] < 0:
            RAAN = 2 * np.pi - RAAN
    else:
        RAAN = 0

    # Argument of perigee
    if n_norm > 1e-8 and e > 1e-8:
        arg_peri = np.arccos(np.dot(n, e_vec) / (n_norm * e))
        if e_vec[2] < 0:
            arg_peri = 2 * np.pi - arg_peri
    else:
        arg_peri = 0

    # True anomaly
    if e > 1e-8:
        nu = np.arccos(np.dot(e_vec, r) / (e * r_norm))
        if np.dot(r, v) < 0:
            nu = 2 * np.pi - nu
    else:
        # For nearly circular orbits, use angle from node vector
        if n_norm > 1e-8:
            nu = np.arccos(np.dot(n, r) / (n_norm * r_norm))
            if r[2] < 0:
                nu = 2 * np.pi - nu
        else:
            nu = 0

    return a, e, inc, RAAN, arg_peri, nu


def two_body_equations(t, state, mu):
    """
    Two-body equations for a central gravitational force.
    state: [rx, ry, rz, vx, vy, vz]
    """
    r = state[0:3]
    v = state[3:6]
    r_norm = np.linalg.norm(r)
    a = -mu * r / r_norm**3
    return np.concatenate((v, a))


if __name__ == "__main__":
    # Initial state for Didymos
    r0 = np.array([-2.39573e8, -2.35661e8, 9.54384e6])  # position in km
    v0 = np.array([12.4732, -9.74427, -0.87661])  # velocity in km/s
    state0 = np.concatenate((r0, v0))

    # Propagation time (seconds)
    tmaxDidymos = 7.0e7
    t_span = (0, tmaxDidymos)
    # Use 1000 time points
    t_eval = np.linspace(0, tmaxDidymos, 1000)

    # Propagate the orbit using ODE solver
    sol = sp.integrate.solve_ivp(
        fun=lambda t, y: two_body_equations(t, y, mu_sun),
        t_span=t_span,
```

```
        y0=state0,
        t_eval=t_eval,
        rtol=1e-9,
        atol=1e-9,
    )

    # Extract the propagated state vectors
    r_sol = sol.y[0:3, :].T  # positions (km)
    v_sol = sol.y[3:6, :].T  # velocities (km/s)

    # Initialize lists for each orbital element
    a_vals = []
    e_vals = []
    inc_vals = []  # in degrees
    RAAN_vals = []  # in degrees
    argp_vals = []  # in degrees
    nu_vals = []  # in degrees

    for r, v in zip(r_sol, v_sol):
        a_i, e_i, inc_i, RAAN_i, argp_i, nu_i = state_to_keplerian(r, v, mu_sun)
        a_vals.append(a_i)
        e_vals.append(e_i)
        inc_vals.append(np.degrees(inc_i))
        RAAN_vals.append(np.degrees(RAAN_i))
        argp_vals.append(np.degrees(argp_i))
        nu_vals.append(np.degrees(nu_i))

    # 3D Orbit Plot
    fig1 = plt.figure(figsize=(10, 8))
    ax1 = fig1.add_subplot(111, projection="3d")
    ax1.plot(r_sol[:, 0], r_sol[:, 1], r_sol[:, 2], "b-", label="Orbit Path")
    ax1.scatter(
        r_sol[0, 0], r_sol[0, 1], r_sol[0, 2], color="green", s=100, label="Start"
    )
    ax1.set_xlabel("X (km)")
    ax1.set_ylabel("Y (km)")
    ax1.set_zlabel("Z (km)")
    ax1.set_title("3D Orbit of Didymos")
    # Set equal axes
    max_range = np.max(np.abs(r_sol))
    ax1.set_xlim([-max_range, max_range])
    ax1.set_ylim([-max_range, max_range])
    ax1.set_zlim([-max_range, max_range])
    ax1.legend()

    # Osculating Orbital Elements Subplots
    fig2, axs = plt.subplots(3, 2, figsize=(14, 12), sharex=True)

    # Semimajor axis
    axs[0, 0].plot(sol.t / 86400, a_vals, "b-")
    axs[0, 0].set_ylabel("a (km)")
    axs[0, 0].set_title("Semimajor Axis")
    axs[0, 0].set_ylim([-max_range, max_range])
    axs[0, 0].grid(True)

    # Eccentricity
    axs[0, 1].plot(sol.t / 86400, e_vals, "r-")
    axs[0, 1].set_ylabel("e")
    axs[0, 1].set_title("Eccentricity")
    axs[0, 1].set_ylim([-max_range, max_range])
    axs[0, 1].grid(True)
```

Problem 2 continued on next page. . .

```python
    # Inclination
    axs[1, 0].plot(sol.t / 86400, inc_vals, "g-")
    axs[1, 0].set_ylabel("i (deg)")
    axs[1, 0].set_title("Inclination")
    axs[1, 0].set_ylim([-max_range, max_range])
    axs[1, 0].grid(True)

    # RAAN
    axs[1, 1].plot(sol.t / 86400, RAAN_vals, "m-")
    axs[1, 1].set_ylabel("RAAN (deg)")
    axs[1, 1].set_title("RAAN")
    axs[1, 1].set_ylim([-max_range, max_range])
    axs[1, 1].grid(True)

    # Argument of Perigee
    axs[2, 0].plot(sol.t / 86400, argp_vals, "c-")
    axs[2, 0].set_ylabel("omega (deg)")
    axs[2, 0].set_title("Argument of Perigee")
    axs[2, 0].set_xlabel("Time (days)")
    axs[2, 0].set_ylim([-max_range, max_range])
    axs[2, 0].grid(True)

    # True Anomaly
    axs[2, 1].plot(sol.t / 86400, nu_vals, "k-")
    axs[2, 1].set_ylabel("nu (deg)")
    axs[2, 1].set_title("True Anomaly")
    axs[2, 1].set_xlabel("Time (days)")
    axs[2, 1].set_ylim([-max_range, max_range])
    axs[2, 1].grid(True)

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

Listing 1: Python code for HW02 P02

Part B

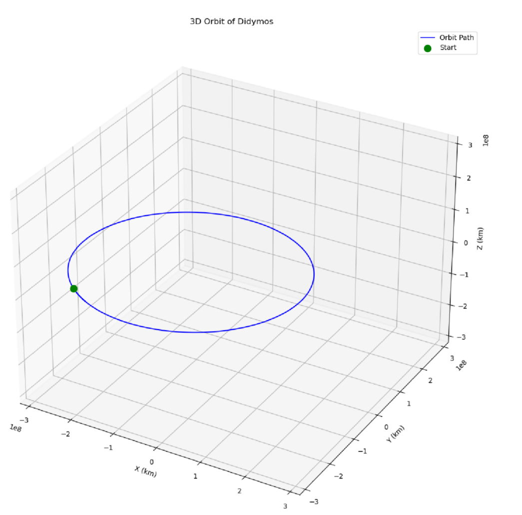

Figure 1: Osculating orbital elements for orbit of Didymos

Part C

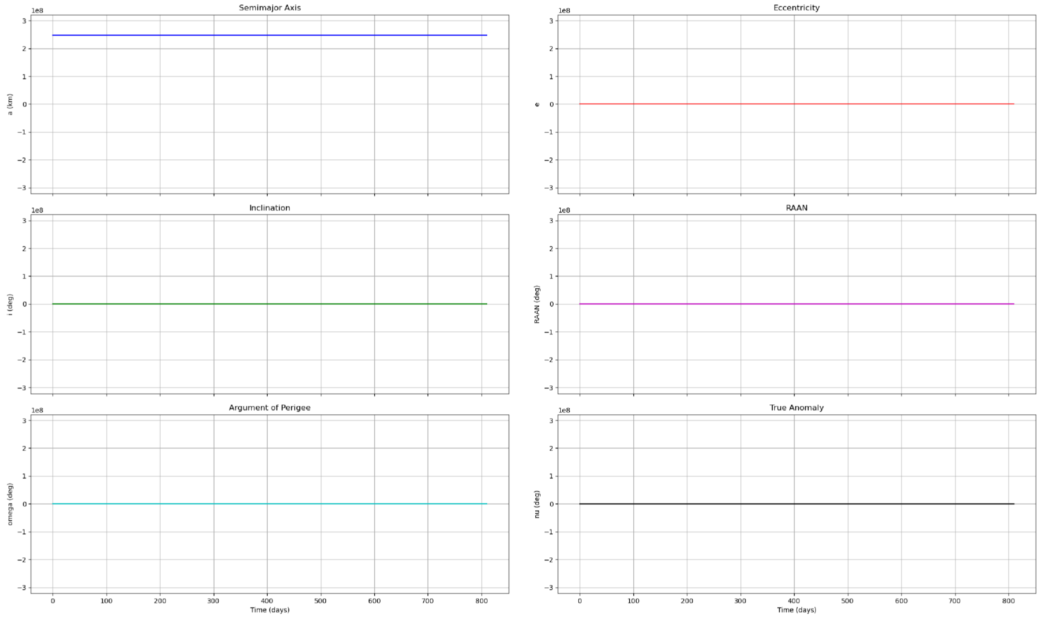

Figure 2: Osculating orbital elements for orbit of Didymos

The 3D plot shows a complete elliptical path. This is consistent with Kepler's first law: objects in a two-body problem follow elliptical orbits around the central body. The starting point is clearly marked, and the overall shape confirms the stability of the orbit under the chosen initial conditions.
In a perfect two-body system, the semimajor axis and eccentricity (which define the size and shape of the ellipse) remain constant. The inclination, RAAN, and argument of perigee, which determine the orbit's orientation, should also remain constant (except for the continuous increase in true anomaly as the body moves along the orbit). The subplots show smooth variations—especially the true anomaly's continuous growth—which confirm that the simulation captures the expected periodic and nearly constant behavior of the other elements. Slight numerical variations can be seen due to the integration method, but overall, the behavior is consistent with the theory.

Problem 3:

Given the following orbit: $a = 2 \times 10^4\,\text{km}, e = 0.4, i = 100°, \Omega = 30°, \omega = 15°, \nu = 15°$

1. Write code to convert from orbital elements to Cartesian coordinates.
2. Propogate the orbit (around Earth) for one period.
3. State the period of the orbit.
4. Plot the orbit in 3D (use equal-length axes).
5. Plot the deviation of the energy as compared to the inital energy ($E_i - E_0$).
6. Plot the osculating orbital elements.

Solution

Part A

See code in Listing 2

Part B

See code in Listing 2

Part C

Orbital period: $7.82\,\text{hr}$

Part D

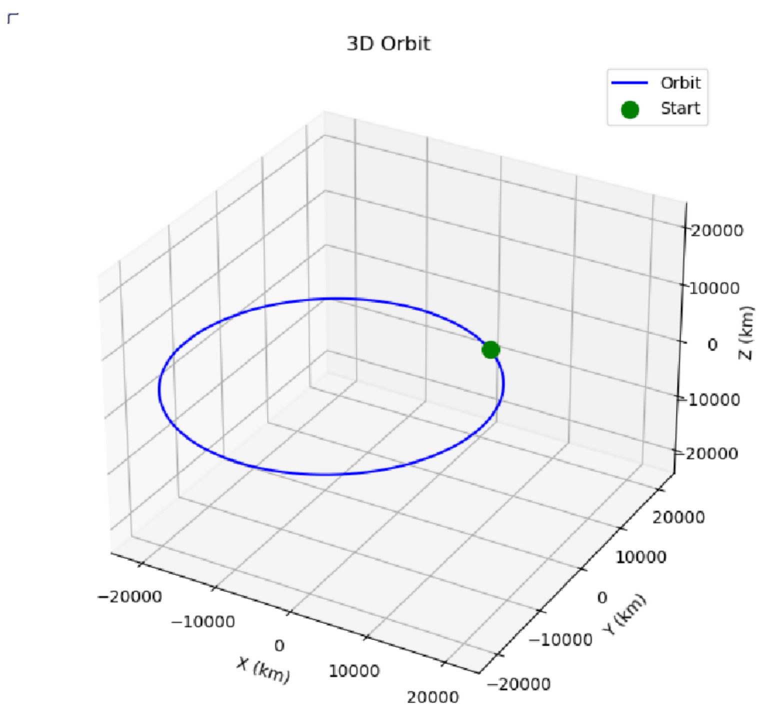

Figure 3: 3D Orbit about Earth

Problem 3 continued on next page. . .

Part E

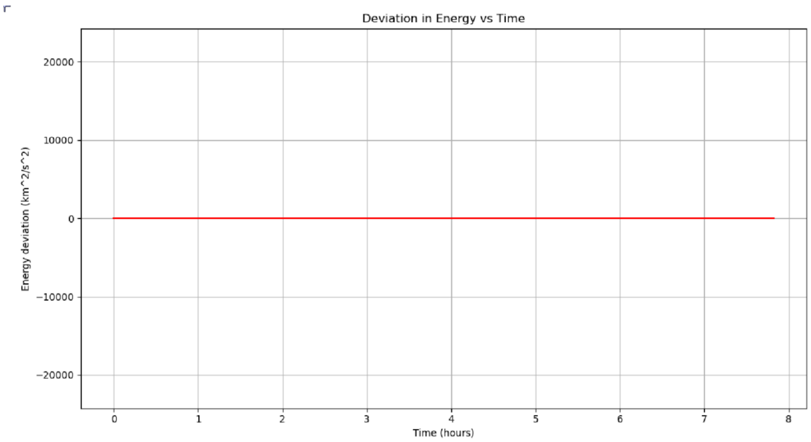

Figure 4: Deviation of Energy w.r.t. Initial Energy ($E_i - E_0$)

Part F

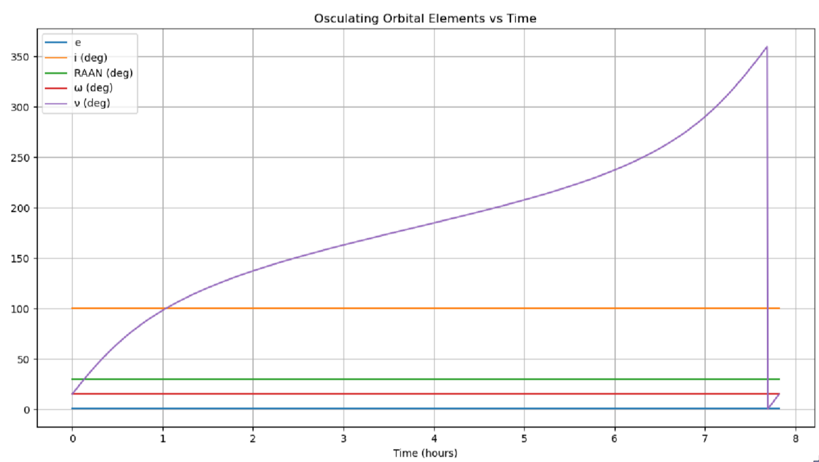

Figure 5: Oscilating Orbital Elements

Code

```python
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

# Earth's gravitational parameter (km^3/s^2)
mu = 398600.0


def keplerian_to_cartesian(a, e, inc, RAAN, arg_perigee, nu, mu):
    """
    Convert orbital elements to Cartesian state vectors (position, velocity)
```

```
    Inputs:
      a           - semimajor axis (km)
      e           - eccentricity (unitless)
      inc         - inclination (rad)
      RAAN        - right ascension of the ascending node (rad)
      arg_perigee - argument of perigee (rad)
      nu          - true anomaly (rad)
      mu          - gravitational parameter (km^3/s^2)
    Returns:
      r_eci, v_eci: position (km) and velocity (km/s) vectors in the ECI frame.
    """
    # Compute the distance (km) from the central body
    r = a * (1 - e**2) / (1 + e * np.cos(nu))

    # Position in the perifocal (PQW) frame
    r_perifocal = np.array([r * np.cos(nu), r * np.sin(nu), 0.0])

    # Parameter p
    p = a * (1 - e**2)
    # Velocity in the perifocal frame
    v_perifocal = np.array(
        [-np.sqrt(mu / p) * np.sin(nu), np.sqrt(mu / p) * (e + np.cos(nu)), 0.0]
    )

    # Rotation matrix from perifocal to ECI frame
    cos_O = np.cos(RAAN)
    sin_O = np.sin(RAAN)
    cos_w = np.cos(arg_perigee)
    sin_w = np.sin(arg_perigee)
    cos_i = np.cos(inc)
    sin_i = np.sin(inc)

    # Transformation matrix (from PQW to ECI)
    R = np.array(
        [
            [
                cos_O * cos_w - sin_O * sin_w * cos_i,
                -cos_O * sin_w - sin_O * cos_w * cos_i,
                sin_O * sin_i,
            ],
            [
                sin_O * cos_w + cos_O * sin_w * cos_i,
                -sin_O * sin_w + cos_O * cos_w * cos_i,
                -cos_O * sin_i,
            ],
            [sin_w * sin_i, cos_w * sin_i, cos_i],
        ]
    )

    # Convert position and velocity into ECI frame
    r_eci = R @ r_perifocal
    v_eci = R @ v_perifocal

    return r_eci, v_eci


def two_body_equations(t, state, mu):
    """
    Equations of motion for the two-body problem.
    state: [rx, ry, rz, vx, vy, vz]
    """
    r = state[0:3]
```

```python
    v = state[3:6]
    r_norm = np.linalg.norm(r)
    # Gravitational acceleration
    a = -mu * r / r_norm**3
    return np.concatenate((v, a))


def state_to_keplerian(r_vec, v_vec, mu):
    """
    Compute orbital elements from state vectors.
    Returns: a, e, inc, RAAN, arg_perigee, nu (all in SI units, angles in rad)
    """
    r = np.array(r_vec)
    v = np.array(v_vec)
    r_norm = np.linalg.norm(r)
    v_norm = np.linalg.norm(v)

    # Specific angular momentum vector and its magnitude
    h = np.cross(r, v)
    h_norm = np.linalg.norm(h)

    # Inclination
    inc = np.arccos(h[2] / h_norm)

    # Node vector (pointing towards ascending node)
    K = np.array([0, 0, 1])
    n = np.cross(K, h)
    n_norm = np.linalg.norm(n)

    # Eccentricity vector
    e_vec = (np.cross(v, h) / mu) - (r / r_norm)
    e = np.linalg.norm(e_vec)

    # Semimajor axis (using vis-viva)
    a = 1 / (2 / r_norm - v_norm**2 / mu)

    # Right ascension of the ascending node (RAAN)
    if n_norm != 0:
        RAAN = np.arccos(n[0] / n_norm)
        if n[1] < 0:
            RAAN = 2 * np.pi - RAAN
    else:
        RAAN = 0

    # Argument of perigee
    if n_norm != 0 and e > 1e-8:
        arg_perigee = np.arccos(np.dot(n, e_vec) / (n_norm * e))
        if e_vec[2] < 0:
            arg_perigee = 2 * np.pi - arg_perigee
    else:
        arg_perigee = 0

    # True anomaly
    if e > 1e-8:
        nu = np.arccos(np.dot(e_vec, r) / (e * r_norm))
        if np.dot(r, v) < 0:
            nu = 2 * np.pi - nu
    else:
        # For circular orbits, true anomaly is undefined; using angle from node vector
        if n_norm != 0:
            nu = np.arccos(np.dot(n, r) / (n_norm * r_norm))
            if r[2] < 0:
```

Problem 3 continued on next page. . .

```
                nu = 2 * np.pi - nu
        else:
            nu = 0

    return a, e, inc, RAAN, arg_perigee, nu


if __name__ == "__main__":
    # Given orbital elements:
    # a in km, e unitless, angles in degrees (convert to radians)
    a = 2e4  # km
    e = 0.4
    inc = np.radians(100)  # inclination
    RAAN = np.radians(30)  # Right Ascension of Ascending Node
    arg_perigee = np.radians(15)  # Argument of perigee
    nu = np.radians(15)  # True anomaly

    # Convert orbital elements to Cartesian state (position and velocity)
    r0, v0 = keplerian_to_cartesian(a, e, inc, RAAN, arg_perigee, nu, mu)
    state0 = np.concatenate((r0, v0))

    # Compute the orbital period using Kepler's third law (T in seconds)
    T = 2 * np.pi * np.sqrt(a**3 / mu)
    print(f"Orbital period: {T/3600:.2f} hours")

    # Time span for propagation (one period)
    t_span = (0, T)
    # Evaluation times (using 1000 sample points)
    t_eval = np.linspace(0, T, 1000)

    # Propagate the orbit using ODE solver
    sol = sp.integrate.solve_ivp(
        fun=lambda t, y: two_body_equations(t, y, mu),
        t_span=t_span,
        y0=state0,
        t_eval=t_eval,
        rtol=1e-9,
        atol=1e-9,
    )

    # Extract position and velocity from the solution
    r_sol = sol.y[0:3, :].T  # shape (N, 3)
    v_sol = sol.y[3:6, :].T  # shape (N, 3)

    # Compute specific mechanical energy at each time step: E = v^2/2 - mu/|r|
    energy = np.array(
        [
            0.5 * np.linalg.norm(v) ** 2 - mu / np.linalg.norm(r)
            for r, v in zip(r_sol, v_sol)
        ]
    )
    E0 = energy[0]
    energy_deviation = energy - E0

    # Osculating Orbital Elements vs Time
    a_vals = []
    e_vals = []
    inc_vals = []
    RAAN_vals = []
    arg_perigee_vals = []
    nu_vals = []
    for r, v in zip(r_sol, v_sol):
```

Problem 3 continued on next page. . .

```
        a_i, e_i, inc_i, RAAN_i, argp_i, nu_i = state_to_keplerian(r, v, mu)
        a_vals.append(a_i)
        e_vals.append(e_i)
        inc_vals.append(np.degrees(inc_i))  # converting to degrees for plotting
        RAAN_vals.append(np.degrees(RAAN_i))
        arg_perigee_vals.append(np.degrees(argp_i))
        nu_vals.append(np.degrees(nu_i))

    # Plotting
    fig = plt.figure(figsize=(14, 10))

    # 3D Orbit plot with equal axes
    ax1 = fig.add_subplot(221, projection="3d")
    ax1.plot(r_sol[:, 0], r_sol[:, 1], r_sol[:, 2], "b-", label="Orbit")
    ax1.scatter(
        r_sol[0, 0],
        r_sol[0, 1],
        r_sol[0, 2],
        color="green",
        marker="o",
        s=100,
        label="Start",
    )
    ax1.set_title("3D Orbit")
    ax1.set_xlabel("X (km)")
    ax1.set_ylabel("Y (km)")
    ax1.set_zlabel("Z (km)")
    # Set equal aspect ratio
    max_range = np.max(np.abs(r_sol))
    ax1.set_xlim([-max_range, max_range])
    ax1.set_ylim([-max_range, max_range])
    ax1.set_zlim([-max_range, max_range])
    ax1.legend()

    # Energy deviation plot
    ax2 = fig.add_subplot(222)
    ax2.plot(sol.t / 3600, energy_deviation, "r-")
    ax2.set_xlabel("Time (hours)")
    ax2.set_ylabel("Energy deviation (km^2/s^2)")
    ax2.set_title("Deviation in Energy vs Time")
    ax2.set_ylim([-max_range, max_range])
    ax2.grid(True)

    # Osculating orbital elements plot (a, e, i, RAAN, arg_perigee, nu)
    ax3 = fig.add_subplot(212)
    # ax3.plot(sol.t / 3600, a_vals, label='a (km)') # skip plotting a, as it is orders of
    magnitude outside range of others
    ax3.plot(sol.t / 3600, e_vals, label="e")
    ax3.plot(sol.t / 3600, inc_vals, label="i (deg)")
    ax3.plot(sol.t / 3600, RAAN_vals, label="RAAN (deg)")
    ax3.plot(sol.t / 3600, arg_perigee_vals, label="omega (deg)")
    ax3.plot(sol.t / 3600, nu_vals, label="nu (deg)")
    ax3.set_xlabel("Time (hours)")
    ax3.set_title("Osculating Orbital Elements vs Time")
    ax3.legend()
    ax3.grid(True)

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

Listing 2: Python code for HW02 P03

Problem 4:

Sketch the following orbits in 2D and 3D. Assume that none of the spacecraft impact Earth.

- In the 2D orbit, label:
  - periapsis
  - angular momentum vector
  - ascending node
  - descending node
  - spacecraft location
  - portion of the orbit in the southern hemisphere
- In the 3D orbit, label:
  - angular momentum vector
  - ascending node
  - periapsis

Spacecraft ID	e	i (°)	Ω (°)	ω (°)	ν (°)
A	0.3	60	30	160	30
B	0.3	60	330	90	10
C	0.5	120	30	30	180

Solution

Part A

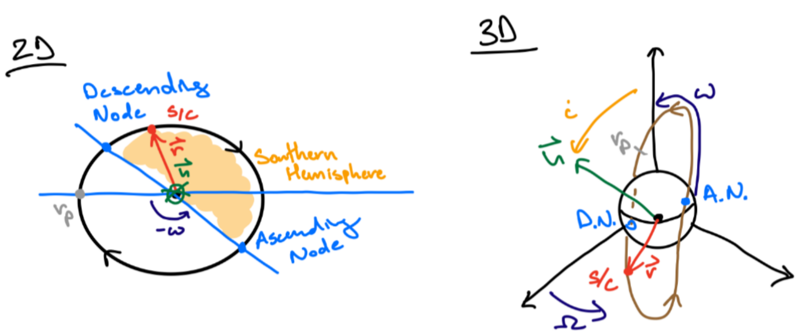

Figure 6: Spacecraft A Orbit in 2D (left) and 3D (right)

Part B

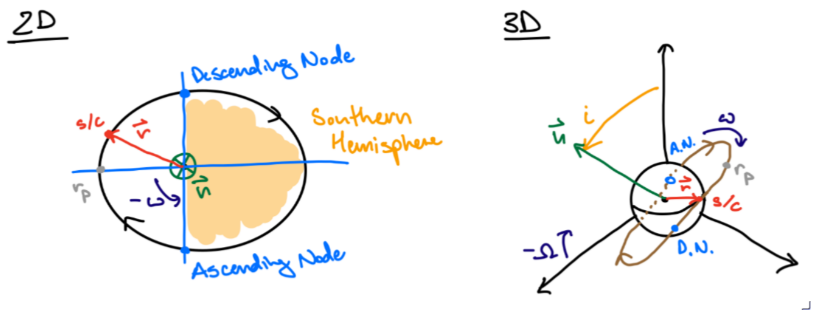

Figure 7: Spacecraft B Orbit in 2D (left) and 3D (right)

Part C

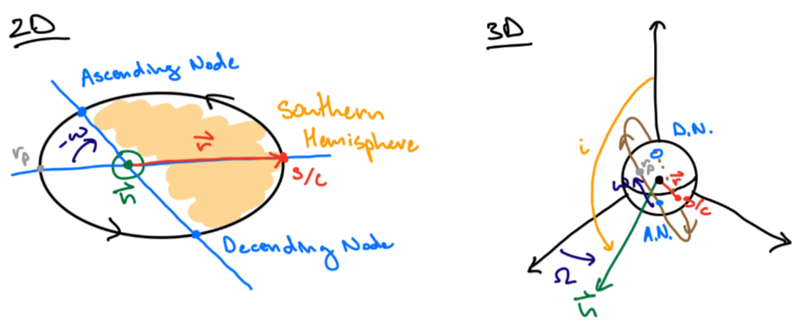

Figure 8: Spacecraft C Orbit in 2D (left) and 3D (right)

Problem 5:

Consider a spacecraft on a hyperbolic trajectory that will fly by Mars. The trajectory's semi-major axis is -11×10^3 km and its eccentricity is 1.8. Calculate:

1. Turn angle
2. Hyperbolic excess speed
3. Miss distance
4. Radius of periapsis of the flyby

Solution

Part A

$$\begin{aligned} a &= -11 \times 10^3 \,\mathrm{km} \\ e &= 1.8 \\ \frac{1}{e} &= \sin \frac{\delta}{2} \\ \delta = 2 \sin^{-1} \frac{1}{e} &= 67.50^\circ \quad \square \end{aligned}$$

Part B

$$\begin{aligned} \mu &= 4.282\,837 \times 10^4 \,\frac{\mathrm{km}^3}{\mathrm{s}^2} \\ v_\infty = \sqrt{\frac{-\mu}{a}} &= 1.973 \,\frac{\mathrm{km}}{\mathrm{s}} \quad \square \end{aligned}$$

Part C

$$\begin{aligned} p = a\left(1 - e^2\right) &= 24\,640 \,\mathrm{km} \\ h = \sqrt{\mu p} &= 32\,485.86 \,\frac{\mathrm{km}^2}{\mathrm{s}} \\ h &= v_\infty \Delta \\ \Delta = \frac{h}{v_\infty} &= 16\,463 \,\mathrm{km} \quad \square \end{aligned}$$

Part D

$$r_p = a\left(1 - e\right) = 8800 \,\mathrm{km} \quad \square$$

Problem 6:

Give the orbital element for an Earth-orbiting spacecraft crossing the $\hat{\boldsymbol{y}}$ axis in a retrograde, equatorial, circular orbit at an altitude of 1 DU. All angles should be given in degrees.

1. What is the semi-major axis (in DU)?
2. Eccentricity?
3. Inclination?
4. Logitude of the ascending node?
5. Argument of periapsis?
6. True anomaly?
7. True longitude at epoch?

Solution

Part A

Circular orbit $\therefore a = 2\,\mathrm{DU}$ $\square$

Part B

Circular orbit $\therefore e = 0$ $\square$

Part C

Retrograde orbit $\therefore i = 180°$ $\square$

Part D

Equatorial orbit $\therefore \Omega = 0$ $\square$

Part E

Circular orbit $\therefore \omega = 0$ $\square$

Part F

Crossing the $\hat{\boldsymbol{y}}$ axis $\therefore \nu = 270°$ $\square$

Part G

Crossing the $\hat{\boldsymbol{y}}$ axis $\therefore L = 270°$ $\square$

Problem 7:

Match the following orbits to the descriptions below:

Spacecraft ID	e	i (°)	Ω (°)	ω (°)	ν (°)
A	1	60	180	160	30
B	2	160	260	90	10
C	0.5	20	210	30	180
D	0.2	90	110	210	270

1. This is a retrograde orbit
2. This spacecraft currently has a positive flight path angle
3. This spacecraft is currently in the southern hemisphere
4. This spacecraft is currently at apoapsis
5. This orbit has a periapsis in the southern hemisphere
6. This orbit has a line of nodes that is colinear with $\hat{x}$

Solution

1. B
2. A, B
3. A, C
4. C
5. D
6. A

Problem 8:

Given an elliptical orbit about the Earth with an eccentricity of 0.3 and a radius of periapsis of 8000 km, calculate the time of flight of the following:

1. From $\nu = 20^\circ \to 30^\circ$
2. From $\nu = 300^\circ \to 20^\circ$

Solution

Part A

$$\begin{aligned}
\nu &= 30^\circ \\
\nu_0 &= 20^\circ \\
k &= 0 \\
\mu &= 3.986 \times 10^5 \, \frac{\text{km}^3}{\text{s}^2} \\
E &= \arccos\left(\frac{e + \cos(\nu)}{1 + e\cos(\nu)}\right) = 0.3883 \, \text{rad} \\
E_0 &= \arccos\left(\frac{e + \cos(\nu_0)}{1 + e\cos(\nu_0)}\right) = 0.2573 \, \text{rad} \\
r_p = a(1-e) \implies a &= \frac{r_p}{1-e} = 11\,428.57 \, \text{km} \\
\Delta T = \sqrt{\frac{a^3}{\mu}} \left(2k\pi + (E - e\sin(E)) - (E_0 - e\sin(E_0))\right) &= 181.35 \, \text{s} \quad \square
\end{aligned}$$

Part B

$$\begin{aligned}
\nu &= 20^\circ \\
\nu_0 &= 300^\circ \\
k &= 1 \\
E &= \arccos\left(\frac{e + \cos(\nu)}{1 + e\cos(\nu)}\right) = 0.2573 \, \text{rad} \\
E_0 = \arccos\left(\frac{e + \cos(\nu_0)}{1 + e\cos(\nu_0)}\right) = -0.801\,47 \, \text{rad} \implies E_0 &= 2\pi + (-0.80147) = 5.4817 \, \text{rad} \\
\Delta T = \sqrt{\frac{a^3}{\mu}} \left(2k\pi + (E - e\sin(E)) - (E_0 - e\sin(E_0))\right) &= 1484.2 \, \text{s} \quad \square
\end{aligned}$$