Problem 1

(a)

In [75]:
```
import numpy as np

U0 = 900 * 1000 / 60 / 60

x_u = -0.1
x_w = -0.05
z_u = -0.1
z_w = -0.2
m_u = -0.1
m_w = -0.1
z_q = -250.1

m_qs = [-0.2, 0.1]

for m in m_qs:
    A = np.array([[x_u, x_w, 0],
                  [z_u, z_w, z_q + U0],
                  [m_u, m_w, m]])

    eigvals = np.linalg.eigvals(A)
    print(f"Eigenvalues when M_q = {m}:", eigvals.tolist())
    print("The system is stable:", np.all(np.real(eigvals) < 0))
    print("")
```

```
Eigenvalues when M_q = -0.2: [-0.32247448713915605, -0.07752551286083821, -0.
10000000000000575]
The system is stable: True

Eigenvalues when M_q = 0.1: [-0.2657535203684179, -0.05989528218678748, 0.125
64880255520522]
The system is stable: False
```

As seen in the python code, when $M_q = -0.2$, the eigenvalues are: $[-0.32247448713915605, -0.07752551286083821, -0.10000000000000575]$ and the system is stable.

Otherwise, when $M_q = 0.1$, the eigenvalues are: $[-0.2657535203684179, -0.05989528218678748, 0.12564880255520522]$ and the system is not stable as the real part of one of the eigenvalues is greater than 0.

(b)

Changing the engines would not be a good idea as the longitudinal stability of the aircraft will be unstable after changing to the new engines. This could lead to erratic behavior of the airplane as it flies.

(c)

Dear CEO,

As an engineer at your company, I am writing this email to warn you not to change the engines to the new, proposed engine. As this new engine needs to be mounted differently, the center of mass of the plane changes and alters the way the airplane pitches. This change results in the plane being unstable as opposed to being stable before and can make the plane dangerous to fly. This is due to the unstability causing unpredictable behavior as the plane flies and can place many lives in the direction of harm. I urge you to keep the old engines, or if the change to the new engines is imperative, then to redesign the plane so that it remains stable.

Problem 2

(a)

In [76]:
```
import sympy as sp

x, y = sp.symbols('x, y')

dx = x * (3 - x - 2*y)
dy = y * (2 - x - y)

x_rate = dx.subs([(x, 1), (y, 0)])
y_rate = dy.subs([(y, 1), (x, 0)])

x_rate, y_rate
```

Out[76]:
```
(2, 1)
```

to find which field is growing faster, we substitute 1 into the equations for each respective variable x, y and hold the other constant at 0 so it remains untouched by the other field.

$$\dot{x}_{untouched} = x * (3 - x)$$
$$\dot{y}_{untouched} = y * (2 - y)$$

and $3 > 2$

The result is that the space sector is growing faster than the aero sector. We can also see this result in the code above where the rate of the space sector is 2000 and the rate of the aero sector is 1000

(b)

In [77]:
```
from sympy import symbols, Eq, solve

x, y = symbols('x y')
x_growth = Eq(x * (3 - x - 2*y), 0)
y_growth = Eq(y * (2 - x - y), 0)

# Find the equilibrium points by solving the system of equations
equilibrium_points = solve((x_growth, y_growth), (x, y))
equilibrium_points
```

Out[77]:
```
[(0, 0), (0, 2), (1, 1), (3, 0)]
```

From above, the coordinates of all the equilibrium points are: $(0, 0), (0, 2), (1, 1), (3, 0)$

(c)

In [78]:
```
Jacobian = Matrix([[dx.diff(x), dx.diff(y)], [dy.diff(x), dy.diff(y)]])
equilibrium_points = solve([dx, dy], (x, y))

# Loop over equilibrium points
for pts in equilibrium_points:
    jacobian_eq = Jacobian.subs([(x, pts[0]), (y, pts[1])])
    jacobian_eq_np = np.array(jacobian_eq).astype(np.float64)
    eigvals = np.linalg.eigvals(jacobian_eq_np)
    print(f"Eigenvalues at equilibrium point {pts}:", eigvals.tolist())
    print("The system is stable at this equilibrium point:", np.all(np.real(ei
    print("")
```

```
Eigenvalues at equilibrium point (0, 0): [3.0, 2.0]
The system is stable at this equilibrium point: False

Eigenvalues at equilibrium point (0, 2): [-2.0, -1.0]
The system is stable at this equilibrium point: True

Eigenvalues at equilibrium point (1, 1): [0.41421356237309515, -2.41421356237
3095]
The system is stable at this equilibrium point: False

Eigenvalues at equilibrium point (3, 0): [-3.0, -1.0]
The system is stable at this equilibrium point: True
```

Using the Jacobian, we can see in the above code that the

equilibrium points: $(0, 0)$ and $(1, 1)$ are unstable,

equilibrium points: $(0, 2)$ and $(3, 0)$ are stable.

(d)

- The lines defined by $3 - x - 2y = 0$ for x and $2 - x - y = 0$ for y are critical for understanding population trajectories.
- The signs of $\dot{x}$ and $\dot{y}$ outside the zero growth lines determine the direction of population changes. For example, if $\dot{x} > 0$ and $\dot{y} < 0$, the population x increases while y decreases.
- System paths in the $x - y$ phase plane show the evolution of the populations. Trajectories do not intersect and tend to move towards or away from equilibrium points.
- The saddle point suggests that one population may grow to dominate the other, potentially leading to the other's decline.
- The absence of closed trajectories around the equilibrium points indicates no stable coexistence cycle between the populations. One population is likely to outcompete the other over time.

Problem 3

(a)

$1g = 9.81 m/s^2$

$$a_c = \frac{v^2}{r}$$

with $v = \omega r$

$$a_c = \frac{(\omega r)^2}{r} = \omega^2 r = \omega^2 \frac{d}{2}$$

$$\omega = \sqrt{\frac{2a_c}{d}} = \sqrt{\frac{2 * 9.81}{500}} = 0.198 \ \frac{rad}{s}$$

(b)

Using the right hand rule, we assume that the spire is spinning counter-clockwise. In order to yaw the spacecraft to the right, we would have to fire the thrusters in the opposite direction of the counter-clockwise spin perpendicular to the z-axis.

(c)

In [80]:
```
g = 9.81
radius = 500 / 2
w1 = 0.198

#original period
T1 = 2 * np.pi / w1

# period at 7g
a_c = 7 * g
w_max = np.sqrt(a_c / radius)
T2 = 2 * np.pi / w_max

alpha = (w_max - w1) / (T1 - T2)

time = (w_max - w1) / alpha / (1/ 30)

print(time)
```

```
592.3415304624111
```

As seen in the above code, the time it takes for the torus to reach 7g is 592.34 seconds

(d)

At 7g acceleration, the concern would likely be the centrifugal force, which is what would be experienced by the astronauts as 'artificial gravity' pushing them against the outer edge of the torus.

Problem 4

(a)

Mach 1.5 is equivalent to:

$$1.5 * 343 = 514.5 \ \ m/s$$

@ sealevel

Comnverting airspeed:

$$900 \ \ km/h * 1000\frac{m}{km} * \frac{1 \ \ h}{3600 \ \ s} = 250 \ \ m/s$$

Radius of the turn @ 9g:

$$r = \frac{v^2}{9g} = \frac{250^2}{9 * 9.81} = 707.894 \ \ m$$

Angular velocity of the turn:

$$\omega = \frac{v}{r} = \frac{250}{707.894} = 0.353$$

Converting to turn rate in degrees per second:

$$\omega = 0.353 * \frac{180}{\pi} = 20.235 \ \ ^\circ/s$$

Time to turn:

$$t_{turn} = \frac{90^\circ}{20.235} = 4.448 \ \ s$$

Time to accelerate:

$$t = \frac{514.5 - 250}{9g} = 3 \ \ s$$

Total time:

$$4.448 + 3 = 7.448 \ \ s$$

(b)

$$L = W \times g$$
$$g = 9 = \sqrt{1 + \tan^2(\theta)}$$
$$g = \frac{1}{9}\sqrt{1 + \tan^2(\theta)}$$
$$L = W = 1.5g$$

$$g = 1$$
$$1 = \frac{1}{9}\sqrt{1 + \tan^2(\theta)}$$
$$\theta = 83.62^\circ$$

The maximum bank angle that can be adopted is 83.62°

Problem 5

(a)

In [123]:
```
w = np.array([0.1, 0.2, 0.1])
I = np.array([600, 500, 500])

tau = 0.1
t_final = 10 * 60

alpha = tau / I
dw = alpha * t_final

dE = (1/2) * I * w**2 - (1/2) * I * (w - dw)**2

print(dE.tolist())
```

```
[3.0000000000000004, 8.400000000000002, 2.4000000000000004]
```

The wheel along the y-axis has the largest change in kinetic energy, so we should spin the wheel associated with the y-axis

(b)

Using Quarternions:

In [124]:
```
def angular_acceleration(I, torque):
    alpha = np.array([0, torque / I[1], 0])
    return alpha

w_body_initial = np.array([0.1, 0.2, 0.1])

def satellite_dynamics(t, y, I, torque):
    q = y[:4]
    w_body = y[4:]

    alpha_body = angular_acceleration(I, torque)

    w_body_dot = alpha_body

    omega_q = np.concatenate(([0], w_body))

    q_dot = 0.5 * np.array([
        q[3] * omega_q[1] - q[2] * omega_q[2] + q[1] * omega_q[3],
        q[0] * omega_q[3] + q[2] * omega_q[1] - q[3] * omega_q[0],
        q[3] * omega_q[0] - q[0] * omega_q[1] + q[1] * omega_q[2],
        -q[1] * omega_q[1] - q[2] * omega_q[2] - q[3] * omega_q[3]
    ])

    return np.concatenate((q_dot, w_body_dot))

y0 = np.concatenate((q_initial, w_body_initial))
t_span = (0, t_final)

solution = solve_ivp(satellite_dynamics, t_span, y0, args=(I, torque_y_axis), 

t_points = np.linspace(0, t_final, 300)
quaternions = solution.sol(t_points)[:4]

euler_angles_series = np.array([R.from_quat(q).as_euler('XYZ', degrees=True) f
```

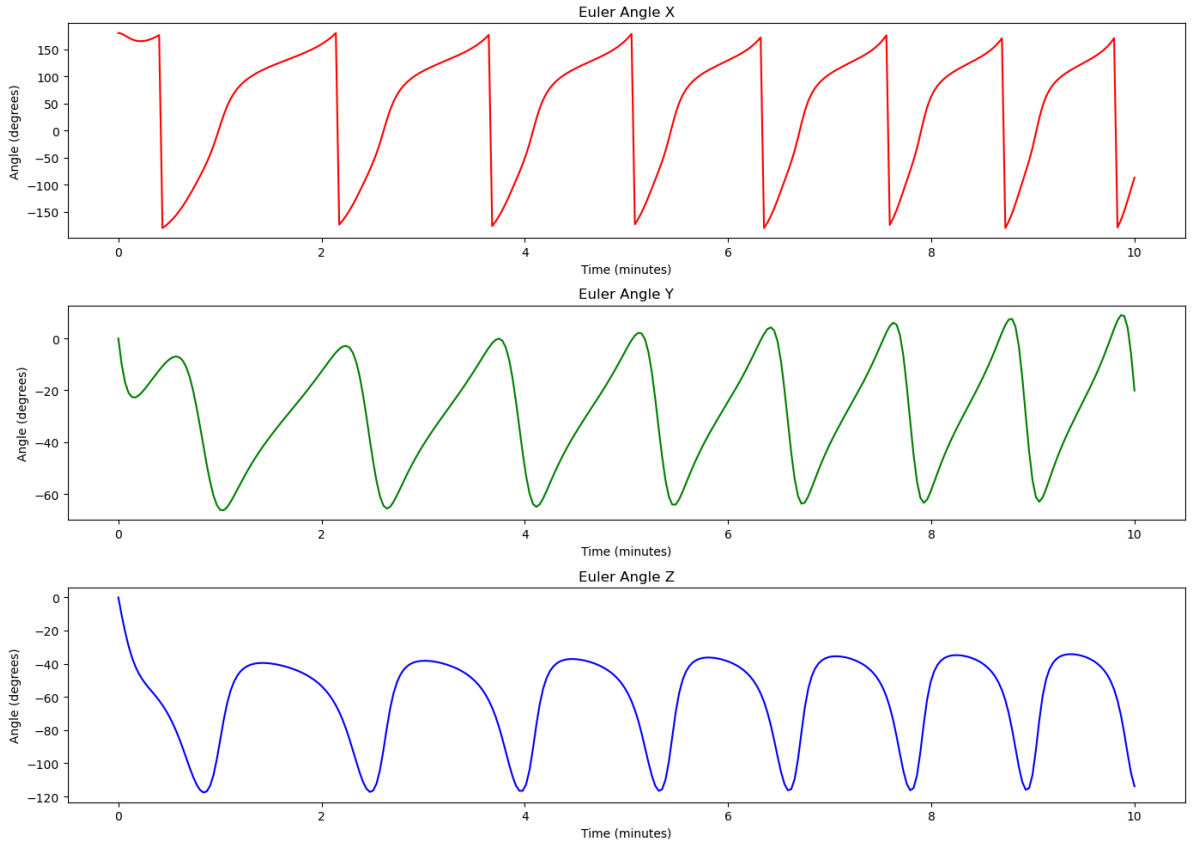
In [125]:
```
plt.figure(figsize=(14, 10))

plt.subplot(3, 1, 1)
plt.plot(t_points / 60, euler_angles_series[:, 0], 'r')
plt.title('Euler Angle X')
plt.xlabel('Time (minutes)')
plt.ylabel('Angle (degrees)')


plt.subplot(3, 1, 2)
plt.plot(t_points / 60, euler_angles_series[:, 1], 'g')
plt.title('Euler Angle Y')
plt.xlabel('Time (minutes)')
plt.ylabel('Angle (degrees)')


plt.subplot(3, 1, 3)
plt.plot(t_points / 60, euler_angles_series[:, 2], 'b')
plt.title('Euler Angle Z')
plt.xlabel('Time (minutes)')
plt.ylabel('Angle (degrees)')

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



In [ ]: