Design of control systems for Aeronautical and Space Vehicles AER4420

Name	Sec/BN#
Sherif Essam Esmail	1/25
Abdelrahman Essam	1/28
Abdelrahman Muhamed	1/30
Muhamed Reffat	2/13
Youssef Ibrahem Ead	2/42

Autopilot Project

22/02/2025

This page is intentionally left blank!

Contents

This page is intentionally left blank!

Chapter 1

AUTOPILOT LITERATURE REVIEW

1.1 Research questions

1.1.1 Definition of Autopilot

An **autopilot** is a control system designed to operate a vehicle, such as an *aircraft, missile, or spacecraft*, with minimal human intervention by automatically regulating its trajectory and stability

1.1.2 Main Objective

The primary objectives of an autopilot system are as follows:

1. **Trajectory Maintenance:** Ensuring that the vehicle follows a predefined flight path by continuously adjusting control surfaces

2. **Stability Augmentation:** Enhancing vehicle stability by compensating for disturbances such as turbulence or atmospheric variations

3. **Workload Reduction:** Minimizing pilot intervention by automating routine control tasks, allowing operators to focus on higher-level decision-making

4. **Operational Efficiency:** Improving fuel consumption and accuracy in navigation, which contributes to safety and effectiveness

1.1.3 Early Autopilot Development [1]

1.1.3.1 Wright Brothers' Innovation (1903):

- Designed an unstable but controllable aircraft, improving maneuverability and reducing susceptibility to atmospheric gusts.

- Early pioneers (Lilienthal, Pilcher, Chanute, Langley) favored inherent stability, making piloting easier but reducing maneuverability.

- The Wright brothers' approach made flying more demanding, pushing the need for automatic pilots (autopilots).

1.1.3.2 Early Autopilot Development:

- "Spiral divergence": A common issue in aircraft where small disturbances cause increasing instability.
- Early autopilots aimed to correct deviations and restore the aircraft's flight attitude.
- **1912: First autopilot tested in *a Curtiss flying boat,* using gyroscopes and servo motors.**
- Sperry Aeroplane Stabilizer (1914): Developed by Sperry Gyroscope Company, led by Dr. E. A. Sperry.
- Won a 50,000-franc safety prize from the Aero Club of France, demonstrated by Lawrence Sperry.
- 1933: Wiley Post & Gyropilot: First pneumatic-hydraulic Gyropilot installed in his plane, Winnie May, Allowed autonomous flight, enabling Post to briefly sleep during an around-the-world flight in 8 days.
- 1947: Fully Automatic Flight: US Air Force C-54 equipped with Sperry A-12 autopilot & Bendix throttle control completed a transatlantic flight without manual control, Program stored on punched cards controlled the flight.
- Challenges in Modern Jets: Early autopilots focused on *maintaining attitude & heading,* Jet aircraft introduced new dynamic stability issues, like *short-period oscillations & Dutch roll*, Human pilots cannot correct rapid oscillations, requiring automatic damping systems, First successful automatic Dutch roll suppression appeared in YB-49 jet bomber.

1.1.4 Inputs and Outputs of an Autopilot System

1.1.4.1 Inputs (Desired/Commanded States)

The autopilot system relies on inputs that define the desired flight parameters. These inputs are typically set by the pilot or a flight management system (FMS). Common inputs include:

- **Altitude**: The target altitude to maintain (e.g., 35,000 feet).
- **Heading**: The desired compass direction (e.g., 270Â° for west).
- **Airspeed**: The target speed through the air (e.g., 250 knots).
- **Vertical Speed**: The desired rate of climb or descent (e.g., 1,000 feet per minute).
- **Pitch and Roll Angles**: The desired aircraft attitude for stability and maneuvering.
- **Navigation Data**: Commands from the FMS or GPS, such as waypoints or flight paths.

1.1.4.2 Outputs (Actuator Commands)

The autopilot generates commands to control the aircraft's actuators, which adjust the control surfaces or engines to achieve the desired states. The primary outputs include:

- **Ailerons**: Control roll by moving up or down to bank the aircraft left or right.
- **Elevators**: Control pitch by moving up or down to raise or lower the nose.
- **Rudder**: Control yaw by moving left or right to adjust the aircraft's heading.
- **Throttle/Thrust**: Adjust engine power to maintain or change airspeed.
- **Spoilers**: Assist in roll control or speed reduction (in some systems).
- **Flaps/Slats**: Adjust lift and drag during specific phases of flight (e.g., landing).

1.1.4.3 Control Loop Process

The autopilot operates as a closed-loop control system, continuously monitoring and adjusting the aircraft's state. The process involves:

1. **Sensing**: The autopilot receives real-time data from sensors (e.g., gyroscopes, accelerometers, GPS, altimeters, airspeed sensors).
2. **Comparison**: The autopilot compares the current state (e.g., actual altitude) with the desired state (e.g., target altitude).
3. **Error Calculation**: The difference (error) between the desired and actual state is calculated.
4. **Control Algorithm**: The autopilot uses control algorithms (e.g., PID controllers) to determine the necessary actuator commands to minimize the error.
5. **Actuation**: The autopilot sends commands to the actuators to adjust the control surfaces or engines, bringing the aircraft closer to the desired state.

1.1.4.4 Example

- **Input**: Desired altitude = 35,000 feet, current altitude = 34,500 feet.
- **Output**: Autopilot commands the elevators to pitch the nose up slightly and adjusts the throttle to increase thrust, causing the aircraft to climb to 35,000 feet.

1.1.5 Human vs Autopilot

While autopilot reduces pilot workload by automating routine tasks, pilots remain essential for:

- Flight Planning and Pre-Flight Checks
- Takeoff and Initial Climb
- Monitoring the Autopilot System
- Handling Emergencies and Malfunctions
- Adjusting Flight Parameters
- Approach and Landing
- Post-Landing and Taxiing

1.1.6 Stability Augmentation and Control Augmentation Systems [2]

Such control systems are known as stability augmentation systems (SASs). If the augmentation system is intended to control the mode and to provide the pilot with a particular type of response to the control inputs, it is known as a control augmentation system(CAS). An example of this is a normal acceleration CAS, in which the pilot's inputs are intended to control the acceleration generated along the negative z-axis. The slow modes (phugoid and spiral) are controllable by a pilot. But since it is undesirable for a pilot to have to pay continuous attention to controlling these modes, an automatic control system is needed to provide "pilot relief." An autopilot is an automatic control system that provides both pilot relief functions and special functions such as automatic landing. The common types of SAS, CAS, and autopilot functions can be listed as follows:

Table 1.1: Comparison of SAS, CAS, and Autopilot Functions

SAS	CAS	Autopilots
Roll damper	Roll rate control	Pitch-attitude hold
Pitch damper	Pitch rate control	Altitude hold
Yaw damper	Normal acceleration	Speed/Mach hold
-	Lateral/directional control	Automatic landing
-	-	Heading hold/VOR hold
-	-	Turn coordination
-	-	Roll angle hold

1.1.7 Onboard Sensors

Table 1.2: Sensors

Sensor	Quantities Measured	Sampling Rates	Sensor Example
GPS	Position (latitude, longitude, altitude), Velocity	1-10 Hz	u-blox NEO-M8N
Gyroscope	Angular velocity (roll, pitch, yaw rates)	100-1000 Hz	InvenSense MPU-6050
Accelerometer	Linear acceleration (x, y, z axes)	100-1000 Hz	STMicroelectronics LSM6DS3
Magnetometer	Magnetic field (heading or yaw)	10-200 Hz	Honeywell HMC5883L
Barometer	Altitude (pressure-based)	10-50 Hz	Bosch BMP280
IMU	Combined motion data	100-2000 Hz	Analog Devices ADIS16470
Airspeed Sensor	Airspeed (velocity relative to air)	100-1000 Hz	Honeywell AMS 5812

1.1.8 Nonmesurable States

the *nonmesurable states* can be estimated from the other states only if the system is *observable*

1.1.8.1 Observability

A system is **observable** if its state can be reconstructed from output measurements over time. This property ensures that unmeasured states can be inferred using a mathematical model.

1.1.8.2 Observability Condition for Linear Systems

Consider a linear time-invariant (LTI) system:

$$\dot{x} = Ax + Bu, \tag{1.1.1}$$

$$y = Cx + Du, \tag{1.1.2}$$

where x is the state vector, u is the input, and y is the measured output.

The system is **observable** if the *observability matrix* has full rank:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{1.1.3}$$

If $\text{rank}(\mathcal{O}) = n$ (where n is the number of states), then all states can be estimated.

1.1.8.3 Observability for Nonlinear Systems

For nonlinear systems, observability is analyzed using the *Observability Gramian* or by applying Lie derivatives. The system dynamics are given by:

$$\dot{x} = f(x, u), \tag{1.1.4}$$

$$y = h(x, u). \tag{1.1.5}$$

The nonlinear observability rank condition requires computing the Lie derivatives of $h(x, u)$ along $f(x, u)$.

1.1.8.4 Observer Design

If a system is observable, an *observer* (state estimator) can be designed to reconstruct non-measured states.

1.1.8.5 Luenberger Observer (Linear Systems)

For an observable LTI system, a Luenberger observer estimates the state:

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x}), \tag{1.1.6}$$

where $\hat{x}$ is the estimated state, and L is the observer gain. The gain L is designed such that the eigenvalues of $(A - LC)$ are stable (negative real parts).

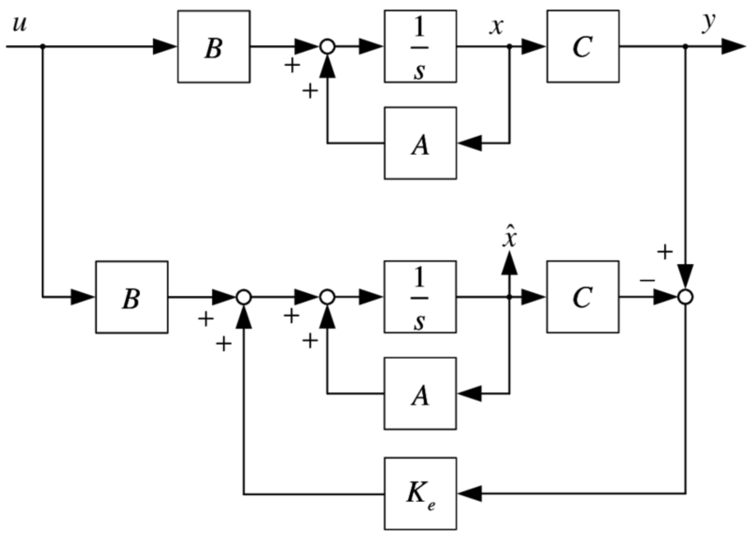


Figure 1.1.1: Observer Design Block Diagram

1.1.9 Fly-By-Wire (FBW) [3]

Fly-By-Wire (FBW) is a modern flight control system that replaces traditional mechanical linkages with electronic signal transmission. In FBW systems, pilot commands are interpreted by flight control computers, which then adjust the control surfaces accordingly.

1.1.9.1 Architecture of Fly-By-Wire Systems

An FBW system consists of the following key components:

1.1.9.2 Flight Control Computers (FCCs)

The FCCs process pilot commands, sensor inputs, and flight control laws to determine the required actuator movements. Modern FBW systems use redundant digital computers to enhance reliability.

1.1.9.3 Sensors and Feedback Systems

FBW systems incorporate gyroscopes, accelerometers, and other sensors to monitor aircraft motion and stability. These inputs are used to adjust control surface deflections dynamically.

1.1.9.4 Actuators

Electronic signals from the FCCs drive electro-mechanical or electro-hydrostatic actuators, which move the control surfaces such as ailerons, elevators, and rudders.

1.1.9.5 Advantages of Fly-By-Wire Systems

FBW systems offer several advantages over traditional mechanical and hydraulic control systems:

- **Weight Reduction:** Eliminates heavy mechanical linkages, reducing overall aircraft weight.
- **Improved Reliability:** Redundant digital computers enhance fault tolerance and system robustness.

- **Enhanced Maneuverability:** FBW allows for real-time adjustments based on sensor feedback, improving stability and responsiveness.
- **Maintenance Reduction:** Unlike hydraulic systems, FBW does not require extensive fluid checks, leak management, or pressure adjustments.
- **Safety Features:** Digital flight control laws can prevent dangerous maneuvers, such as stalls or excessive structural loads.

1.1.10 Autopilot Software VS Hardware [3]

Table 1.3: Comparison of Software and Hardware Autopilot Systems

Category	Software Autopilot		Hardware Autopilot	
Definition	Flight control algorithms and programs running on an autopilot system.		The physical computing unit executing autopilot software.	
Function	Manages UAV stabilization, navigation, control, and autonomous operations.		Provides processing power, sensor data acquisition, and control execution.	
Components	Source code (C/C++, Python), control algorithms, mission planning tools.		Microcontroller/processor, IMU (gyroscope, accelerometer, magnetometer), barometer, GPS, communication modules.	
Customization	Can be modified, updated, and adapted for different vehicles.		Limited by hardware specifications but can support different software.	
Examples	**ArduPilot**	(Runs on various hardware)	**PX4 Pixhawk**	(Hardware running PX4 software)
	PX4	(Open-source, used in different autopilot boards)	**ArduPilot Mega (APM)**	(Hardware for ArduPilot)
			MicroPilot MP2x28xp	(Commercial UAV autopilot)
	MicroPilot Horizon	(Ground control software)	**DJI WooKong-M**	(Proprietary autopilot hardware)
Availability	Open-source (ArduPilot, PX4) or proprietary (MicroPilot Horizon).		Open-hardware (PX4 Pixhawk) or proprietary (DJI WooKong, MicroPilot MP2x28xp).	
Development	Continuously updated with new features and improvements.		Hardware remains the same but can support firmware updates.	

1.2 Flight Mechanics Review

1.2.1 a) General Rigid Body Dynamics Equations in 3D (12 Equations)

A rigid body in 3D is fully described by 12 equations, split equally into 6 kinetic (dynamics) and 6 kinematic (geometric) equations.

Kinetic Equations (Dynamics)

These equations govern the forces and moments acting on the body.

1.2.1.1 Translational Dynamics (Newton's Second Law)

$$\dot{u} = (F_{\mathrm{x}}/m) + r \cdot v - q \cdot w \tag{1.2.1}$$

$$\dot{v} = (F_y/m) + p \cdot w - r \cdot u \tag{1.2.2}$$

$$\dot{w} = (Fz/m) + q \cdot u - p \cdot v \tag{1.2.3}$$

Where:

- $u, v, w =$ Velocity components along the body $x-, y-$, and z-axes
- $F_{\mathrm{x}}, F_y, Fz =$ External force components along the body axes
- $m =$ Mass of the body
- $p, q, r =$ Angular velocity components about the body x -, y -, and z -axes

1.2.1.2 Rotational Dynamics (Euler's Equations)

$$\dot{p} = (1/I_{\mathrm{x}}) \cdot [M_{\mathrm{x}} + (I_y - Iz) \cdot q \cdot r] \tag{1.2.4}$$

$$\dot{q} = (1/I_{\mathrm{y}}) \cdot [M_y + (Iz - I_{\mathrm{x}}) \cdot r \cdot p] \tag{1.2.5}$$

$$\dot{r} = (1/Iz) \cdot [Mz + (I_{\mathrm{x}} - I_y) \cdot p \cdot q] \tag{1.2.6}$$

Where:

- $I_{\mathrm{x}}, I_y, Iz =$ Moments of inertia about the body axes
- $M_{\mathrm{x}}, M_y, Mz =$ External moment components about the body axes

Kinematic Equations (Geometry)

These equations relate the velocities and angular rates to the changes in position and orientation.

1.2.1.3 Translational Kinematics (Position Updates)

$$\dot{\mathrm{x}} = u \cdot \cos\theta \cdot \cos\psi + v \cdot (\sin\varphi \cdot \sin\theta \cdot \cos\psi - \cos\varphi \cdot \sin\psi) + w \cdot (\cos\varphi \cdot \sin\theta \cdot \cos\psi + \sin\varphi \cdot \sin\psi) \quad (1.2.7)$$

$$\dot{\mathrm{y}} = u \cdot \cos\theta \cdot \sin\psi + v \cdot (\sin\varphi \cdot \sin\theta \cdot \sin\psi + \cos\varphi \cdot \cos\psi) + w \cdot (\cos\varphi \cdot \sin\theta \cdot \sin\psi - \sin\varphi \cdot \cos\psi) \quad (1.2.8)$$

$$\dot{\mathrm{z}} = -u \cdot \sin\theta + v \cdot \sin\varphi \cdot \cos\theta + w \cdot \cos\varphi \cdot \cos\theta \quad (1.2.9)$$

Where:

- $x, y, z =$ Inertial (earth-fixed) coordinates
- $\varphi, \theta, \psi =$ Euler angles (roll, pitch, yaw)

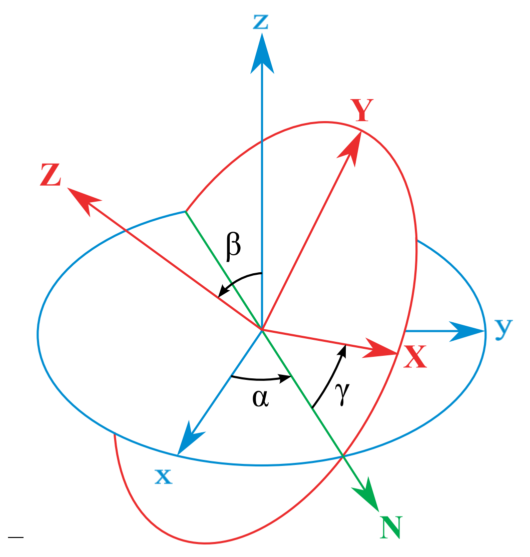


Figure 1.2.1: Euler Angles

1.2.1.4 Rotational Kinematics (Euler Angle Rates)

$$\dot{\varphi} = p + q \cdot \sin\varphi \cdot \tan\theta + r \cdot \cos\varphi \cdot \tan\theta \quad (1.2.10)$$

$$\theta = q \cdot \cos\varphi - r \cdot \sin\varphi \quad (1.2.11)$$

$$\psi = (q \cdot \sin\varphi + r \cdot \cos\varphi)/\cos\theta \quad (1.2.12)$$

These 12 equations together comprehensively describe the motion of a rigid body in 3D space.

1.2.2 b) Classification into Kinetics and Kinematics Kinetic Equations (6 in total):

- Equations (1) - (3): Translational dynamics

- Equations (4) - (6): Rotational dynamics

Kinematic Equations (6 in total):

- Equations (7) - (9): Relate body velocities (expressed in body frame) to inertial position changes
- Equations (10) - (12): Relate body angular rates to Euler angle rates

1.2.3 c) Additional Equations for Fixed-Wing Airplane Equations of Motion (EOM)

To specialize the general RBD equations for a fixed-wing airplane, the following are added: •

1.2.3.1 Aerodynamic Force and Moment Models:

Aerodynamic forces are expressed as functions of dynamic pressure, wing area, and non-dimensional coefficients (e.g., Cx, Cy, Cz). Similarly, aerodynamic moments (roll L, pitch M, yaw N) are modeled using coefficients that depend on angle of attack, sideslip, and control surface deflections.

1.2.3.2 - Gravitational Force Projection:

The weight of the airplane is resolved along the body axes via the transformation from the inertial (or earthfixed) frame.

1.2.3.3 Velocity-to-Angle Relationships:

Equations that relate the components of velocity in the body frame to the airspeed, angle of attack (α), and sideslip angle (β).

- Control Input Effects:

Additional relations that quantify how control surface deflections affect the aerodynamic forces and moments.

1.2.4 d) Assumptions in Deriving Airplane Equations of Motion.

Common assumptions include:

- Rigid Body Approximation: The aircraft is assumed not to deform.
- Constant Mass and Inertia: Effects of fuel burn or payload shifts are neglected.
- Quasi-Steady Aerodynamics: Aerodynamic forces and moments are assumed to respond instantaneously to changes in flight conditions.
- Small Perturbations: Linearization is often performed around a trim condition.
- Neglection of High-Order Effects: Compressibility, viscous effects, and higher-order nonlinearities are typically ignored in preliminary analyses.
- Uniform Atmospheric Conditions: The air is assumed to have steady and uniform properties.

1.2.5 e) Mathematical Classification of the Airplane EOM.

Table 1.4: Classification of the Airplane EOM

Order	The equations are represented as first-order differential equations in state-space form.
Type	They are Ordinary Differential Equations (ODEs), with time being the independent variable.
Linearity	The full equations are nonlinear, although linearization is common near steady flight conditions.
Coupling	The equations are generally coupled, as translational and rotational dynamics interact. Under certain assumptions (such as decoupling of longitudinal and lateral-directional dynamics), the equations can be simplified to an uncoupled form.

1.2.6 f) Difference Between Body Axes and Earth (Inertial) Axes.

1.2.6.1 - Body Axes:

A coordinate system fixed to the airplane, typically defined as:

- x - axis: along the fuselage (forward),
- y - axis: towards the right wing,
- z-axis: downward.

The body axes rotate with the aircraft.

1.2.6.2 Earth (Inertial) Axes:

A fixed or quasi-fixed coordinate system relative to the Earth, often defined as North-East-Down (NED) or a similar scheme, serving as an absolute reference frame.

1.2.7 g) Difference Between Pitch Angle (θ) vs. Angle of Attack (α) and Sideslip Angle (β) vs. Heading Angle (ψ).

1.2.7.1 Pitch Angle (θ) vs. Angle of Attack (α):

The pitch angle θ is the orientation of the aircraft's longitudinal axis relative to the horizon, while the angle of attack α is the angle between the chord line (or x-axis) and the oncoming airflow. In steady, coordinated flight these angles are related, but during maneuvers or in the presence of wind, they can differ significantly.

1.2.7.2 Sideslip Angle (β) vs. Heading Angle (ψ):

The sideslip angle β is the angle between the aircraft's longitudinal axis and the relative wind, whereas the heading angle ψ is the navigational direction of the aircraft relative to a fixed reference (such as geographic North).

1.2.8 h) Attitude Representations: Advantages and Disadvantages

1.2.8.1 - Euler Angles:

- Advantages: Intuitive (roll, pitch, yaw) and straightforward.
- Disadvantages: Suffer from singularities (gimbal lock) when, for example, the pitch angle approaches ± 90.

1.2.8.2 - Direction Cosine Matrix (DCM):

- Advantages: No singularity issues and provides a full transformation matrix.
- Disadvantages: Requires 9 elements along with orthogonality constraints, which can complicate numerical implementation.

1.2.8.3 - Quaternions:

- Advantages: Compact (4 parameters), computationally efficient, and free from singularities.
- Disadvantages: Less intuitive and involve a double-cover issue (i.e., q and -q represent the same orientation).
- Axis-Angle Representation:
- Advantages: Provides a clear geometric interpretation (rotation by a specific angle about a fixed axis).
- Disadvantages: Less practical for sequential rotations or for numerical integration when rotations are small.

1.3 Numerical solution of ODEs

1.3.1 Some of the numerical solving algorithms for ODEs

Numerical solutions are approximate solutions to the differential equation where the analytical solution is hard to get or even not available as the case in airplane equations. The variation of the algorithms is due to the need for different approaches for solving the equations like: Accuracy requirements, stability considerations, computational efficiency, step size adaptability, and system constraints.

Reason for the Variation of Numerical Algorithms. The wide variety of numerical algorithms for solving Ordinary Differential Equations (ODEs) exists because different problems have unique characteristics that require specialized approaches. The main factors influencing algorithm selection include:

1. Accuracy Requirements - Some applications demand high precision (e.g., spacecraft navigation), while others prioritize speed (e.g., real-time control systems).
2. Stability Considerations - Certain ODEs, especially stiff systems (e.g., chemical reactions, flight dynamics at high speeds), require implicit methods for stable solutions.
3. Computational Efficiency - Simpler methods (e.g., Euler's) require fewer calculations but may need smaller time steps, while higher-order methods (e.g., Runge-Kutta) balance accuracy and efficiency.
4. Step Size Adaptability - Some methods (e.g., RK45) adjust the step size automatically to handle rapidly changing or smooth regions efficiently.

5. System Constraints - Real-time applications (e.g., autopilot, robotics) need algorithms that work under computational and time constraints.

1.3.1.1 Benefits of Algorithm Variation

Optimized Performance for Different Applications

- Euler's Method is useful for quick approximations.
- RK4 provides high accuracy for aerospace simulations.
- Implicit methods (e.g., Crank-Nicolson) are better for stiff problems like structural vibrations.

Balancing Accuracy & Speed

- High-order methods (RK4, RK45) are more precise but computationally expensive.
- Lower-order methods (Euler, Adams-Bashforth) are faster but may require smaller steps.

Handling Stiff vs. Non-Stiff Systems

- Stiff ODEs (e.g., high-speed aerodynamic models) require implicit methods for stability.
- Non-stiff ODEs (e.g., basic motion equations) can use explicit methods.

Adaptability to Problem Complexity

- Verlet integration conserves energy in orbital mechanics.
- Predictor-Corrector methods improve accuracy in long-duration flight simulations.

Real-Time Applications

- Some algorithms (e.g., fixed-step RK4) are used in autopilot systems to meet real-time constraints.
- Adaptive methods (RK45) are useful in simulations but may be too slow for real-time control

Table 1.5: Summary of Numerical Algorithms for Solving ODEs

Method	Order	Type	Pros	Cons
Euler's Method	1st	Explicit	Simple, fast	Low accuracy, unstable for large steps
RK4	4th	Explicit	High accuracy, widely used	Computationally expensive
RK45	Adaptive	Explicit	Efficient, variable step size	More complex
Adams-Bashforth	Variable	Explicit	Efficient, reuses previous values	Requires initialization
Adams-Moulton	Variable	Implicit	Stable, good for stiff problems	Computational overhead
Backward Euler	1st	Implicit	Stable for stiff ODEs	Requires solving nonlinear equations
Crank-Nicolson	2nd	Implicit	Stable, accurate	More complex than Backward Euler
Verlet Integration	2nd	Explicit	Energy-conserving	Requires two previous states

- Explicit Methods: Easier to implement but may require small step sizes for stability.
- Implicit Methods: More stable, especially for stiff problems, but require solving equations at each step.
- Adaptive Methods: Adjust step size automatically for better efficiency.
- Specialized Methods: Used for specific applications like orbital mechanics (Verlet) or flight control (Predictor-Corrector).

Choosing one algorithm for solving the Airplanes EOM, clearly state the (Initial conditions needed, Inputs needed in each iteration, and Outputs calculated in each iteration) The equation of motion of an airplane is derived from the rigid body motion equations. Those equations could be divided into five (or four) sets of equations:

Force equations

$$X - mgS_\theta = m(\dot{u} + qw - rv)$$
$$Y + mgC_\theta S_\Phi = m(\dot{v} + ru - pw)$$
$$Z + mgC_\theta C_\Phi = m(\dot{w} + pv - qu)$$

Moment equations

$$L = I_x\dot{p} - I_{xz}\dot{r} + qr\left(I_z - I_y\right) - I_{xz}pq$$
$$M = I_y\dot{q} + rq\left(I_x - I_z\right) + I_{xz}\left(p^2 - r^2\right)$$
$$N = -I_{xz}\dot{p} + I_z\dot{r} + pq\left(I_y - I_x\right) + I_{xz}qr$$

Angular velocity

$$p = \dot{\Phi} - \dot{\psi}S_\theta$$
$$q = \dot{\theta}C_\Phi + \dot{\psi}C_\theta S_\Phi$$
$$r = \dot{\psi}C_\theta C_\Phi - \dot{\theta}S_\Phi$$

Velocity equations

$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dx}{dt} \\ \frac{dz}{dt} \end{bmatrix} = \begin{bmatrix} C_\theta C_\psi & S_\Phi S_\theta C_\psi - C_\Phi S_\psi & C_\psi S_\theta C_\psi + S_\Phi S_\psi \\ C_\theta S_\psi & S_\Phi S_\theta S_\psi + C_\Phi C_\psi & C_\Phi S_\theta S_\psi - S_\Phi C_\psi \\ -S_\theta & S_\Phi C_\theta & C_\phi C_\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

We need to get every derivative W.R.T. time so we will apply the small disturbance approximation where we could swap every variable with an initial known value and the change in this value. This approximation is only valid when the changes in those variables (states) are very small which could be seen almost at the cruise conditions throughout the journey of an aircraft.
The variables could be replaced as follows:

$$u = u_0 + \Delta u$$
$$v = v_0 + \Delta v$$
$$w = w_0 + \Delta w$$
$$p = p_0 + \Delta p$$
$$q = q_0 + \Delta q$$
$$r = r_0 + \Delta r$$
$$X = X_0 + \Delta X$$
$$Y = Y_0 + \Delta Y$$
$$Z = Z_0 + \Delta Z$$
$$M = M_0 + \Delta M$$
$$N = N_0 + \Delta N$$
$$L = L_0 + \Delta L$$
$$\delta = \delta_0 + \Delta\delta$$

Flight conditions are taken to be symmetric to simplify the equation to be:

$$\left.\begin{array}{rc} \Delta X = & \frac{\partial X}{\partial u}\Delta u + \frac{\partial X}{\partial w}\Delta w \\ \Delta Y = & \frac{\partial Y}{\partial v}\Delta v + \frac{\partial Y}{\partial p}\Delta p + \frac{\partial Y}{\partial r}\Delta r \\ \Delta Z = & \frac{\partial Z}{\partial u}\Delta u + \frac{\partial Z}{\partial w}\Delta w + \frac{\partial Z}{\partial \dot{w}}\Delta \dot{w} + \frac{\partial Z}{\partial q}\Delta q \end{array}\right\}$$

$$\Delta L = \frac{\partial L}{\partial v}\Delta v + \frac{\partial L}{\partial p}\Delta p + \frac{\partial L}{\partial r}\Delta r$$

$$\Delta M = \frac{\partial M}{\partial u}\Delta u + \frac{\partial M}{\partial w}\Delta w + \frac{\partial M}{\partial \dot{w}}\Delta \dot{w} + \frac{\partial M}{\partial q}\Delta q$$

$$\Delta N = \frac{\partial N}{\partial v}\Delta v + \frac{\partial N}{\partial p}\Delta p + \frac{\partial N}{\partial r}\Delta r$$

We will choose RK4 as our algorithm for solving the EOM, and here are the ICs, BCs, and the outputs of each iteration:

Initial conditions - Before starting the numerical integration, we need the initial state of the aircraft, which typically includes:

- Position: (initial location in a given coordinate system)
- Velocity: (initial velocity components in body frame)
- Attitude (Orientation): (initial roll, pitch, and yaw angles)
- Angular Velocity: (initial roll, pitch, and yaw rates)
- Mass & Inertia Properties: (aircraft mass and moments of inertia)

Input needed in each iteration - At each time step, the numerical algorithm needs updated information to compute the next state:

- Aerodynamic Forces & Moments:
  1. Lift, Drag, and Side Forces
  2. Moments about body axes
- Control Inputs (from Pilot or Autopilot):
  1. Throttle Setting
  2. Elevator Deflection
  3. Aileron Deflection
  4. Rudder Deflection
- Environmental Conditions:
  1. Wind Velocity Components
  2. Air Density
  3. Gravity

Outputs Calculated in Each Iteration -The numerical algorithm updates and computes the next state of the aircraft:

- Updated position
- Updated velocity components
- Updated attitude (orientation)
- Updated angular velocity
- New forces & moments

1.3.2 RK4

Solving Problem1 by RK4.

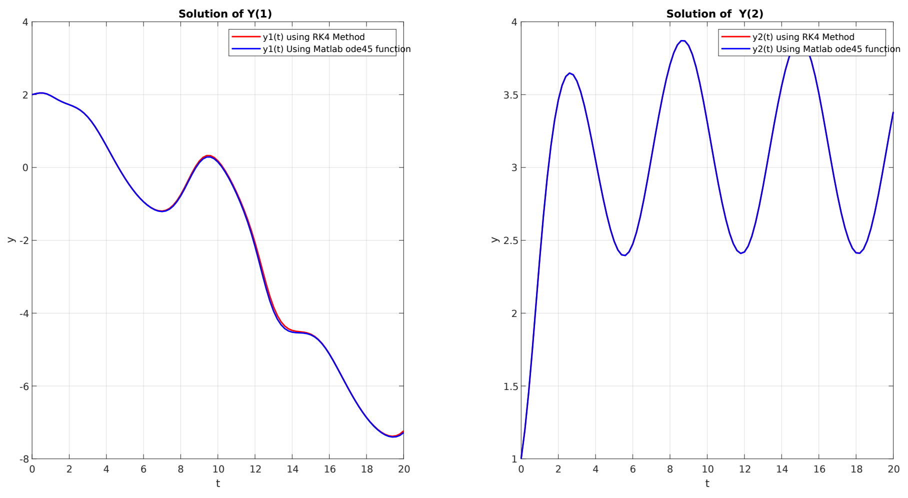


Figure 1.3.1: Our Code VS MATLAB

This page is intentionally left blank!

Chapter 2

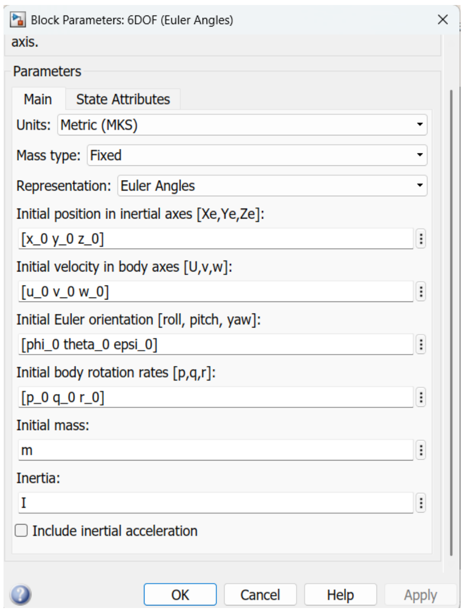
AIRPLANE SIMULATOR PART I (RK4)

2.1 Simulink Model

To Verify and Validate the results obtained from the code that solved RBD equations using "Runge-Kutta” 4th order method. The code results was compared with the results from Simulink model that use the 6 DOF Euler angles block to simulate the RGB equations.The mass of the body was assumed to be constant and the properties defined for the 6 DOF Euler angles block is:

- Initial position in inertial axes (X_0, Y_0, Z_0)
- Initial Velocity in body axes (u_0, v_0, w_0)
- Inertial Euler Orientation $(\phi_0, \theta_0, \psi_0)$
- Initial body rotation rates (p_0, q_0, r_0)
- mass
- Inertia (3X3 Matrix)

These values were defined in the code then imported from the workspace.


Block Parameters: 6DOF (Euler Angles)

axis.

Parameters

Main | State Attributes

Units: Metric (MKS)

Mass type: Fixed

Representation: Euler Angles

Initial position in inertial axes [Xe,Ye,Ze]:
[x_0 y_0 z_0]

Initial velocity in body axes [U,v,w]:
[u_0 v_0 w_0]

Initial Euler orientation [roll, pitch, yaw]:
[phi_0 theta_0 epsi_0]

Initial body rotation rates [p,q,r]:
[p_0 q_0 r_0]

Initial mass:
m

Inertia:
I

☐ Include inertial acceleration

Figure 2.1.1: Input to the 6 DOF Euler angles block

$$mass = 11\ kg$$

$$\boldsymbol{I} = \begin{bmatrix} 1 & -2 & -1 \\ -2 & 5 & -4 \\ -1 & -4 & 0.2 \end{bmatrix} kg.m^2$$

$$[u, v, w, p, q, r, \phi, \theta, \psi, x, y, z]_{t=0} =$$

$$\left[10, 2, 0, 2 * \frac{\pi}{180}, 1 * \frac{\pi}{180}, 0 * \frac{\pi}{180}, 20 * \frac{\pi}{180}, 15 * \frac{\pi}{180}, 30 * \frac{\pi}{180}, 2, 4, 7\right]$$

Figure 2.1.2: The Values Used for Euler block properties

The input to the 6 DOF Euler angles block is Forces and Moments which has values of:

Forces $= [\ 2 \quad 8 \quad 3\ ]N$

Moments $= [\ 14 \quad 20 \quad 7\ ]N.m$

The simulation time was 25 s

Then the output values of the model was exported back to the code to be compared with the output values of the "Runge-Kutta" 4th order method code.

The output values exported back were:

- position in inertial axes (X, Y, Z)
- Initial Velocity in body axes (u, v, w)
- Inertial Euler Orientation (ϕ, θ, ψ)
- Initial body rotation rates (p, q, r)

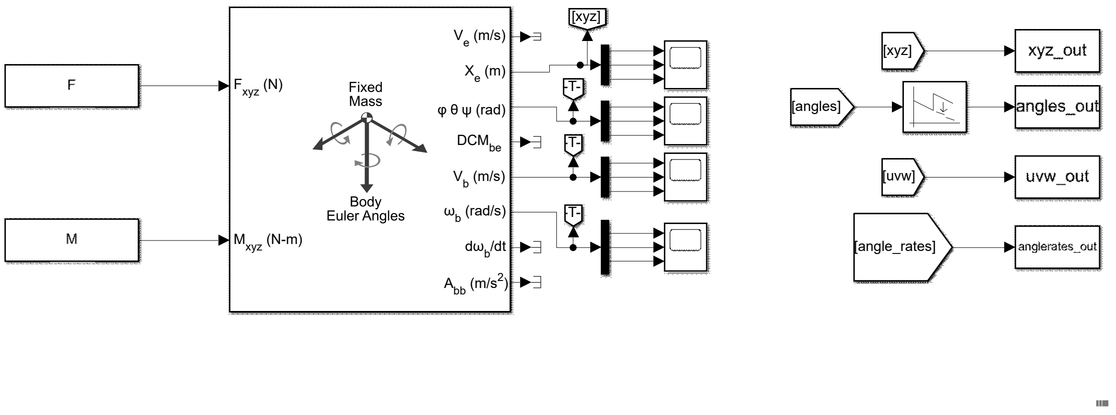
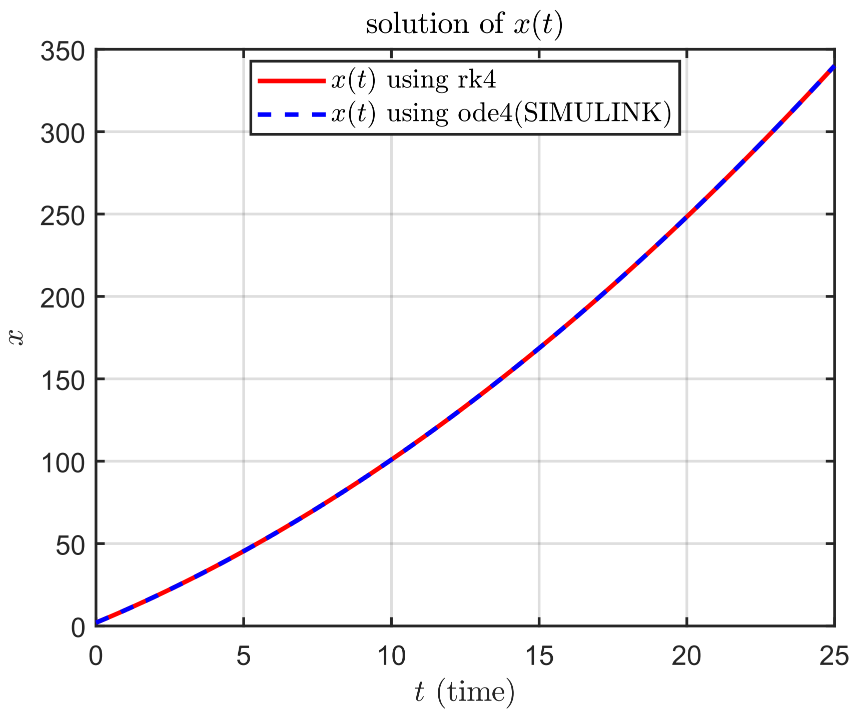
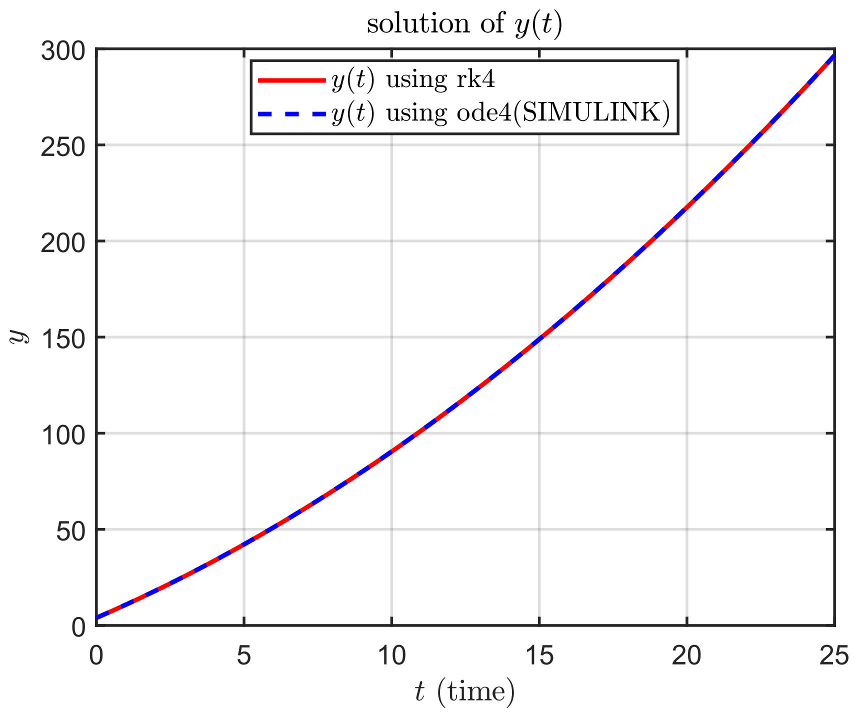


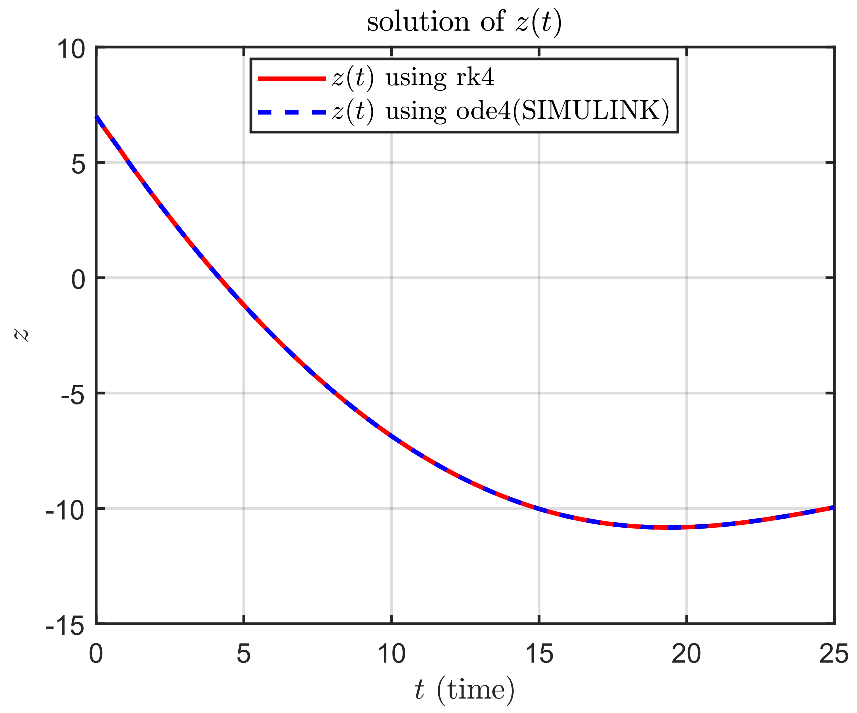
Figure 2.1.3: Simulink Model Used

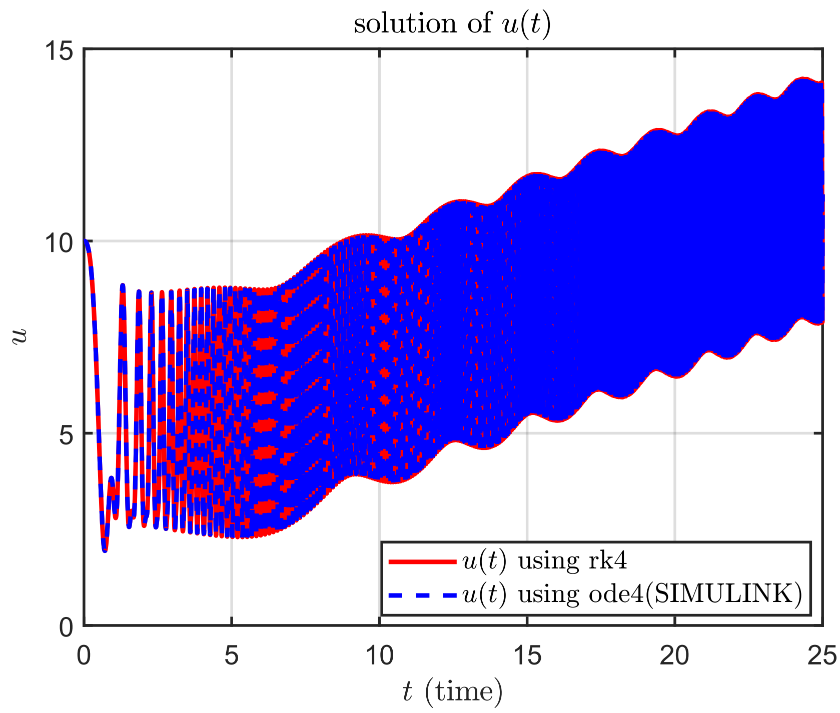
2.2 Comparing Results

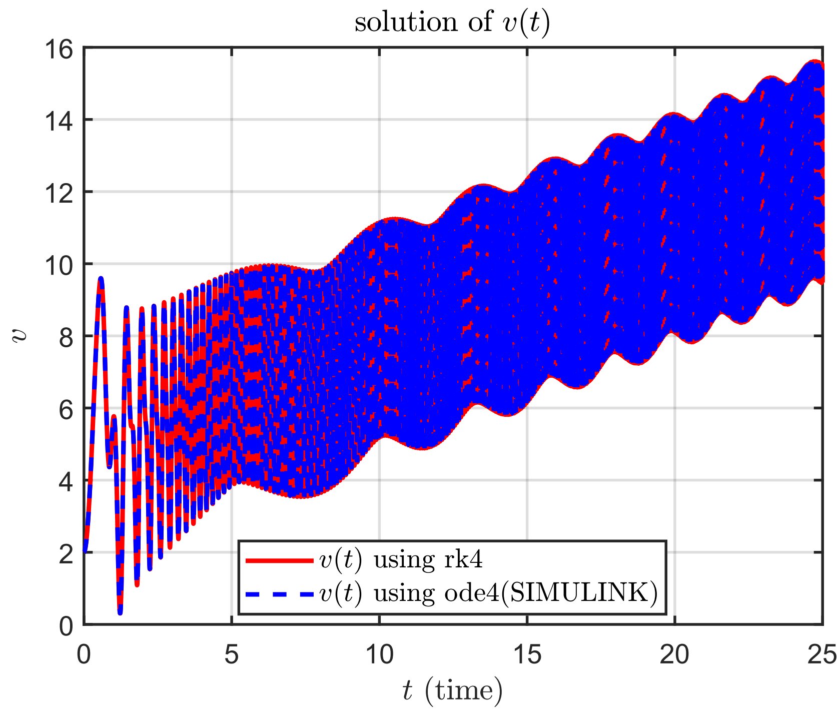
The figures comparing the values coming out of Simulink model and AIRPLANE SIMULATOR PART I code output are as follows:

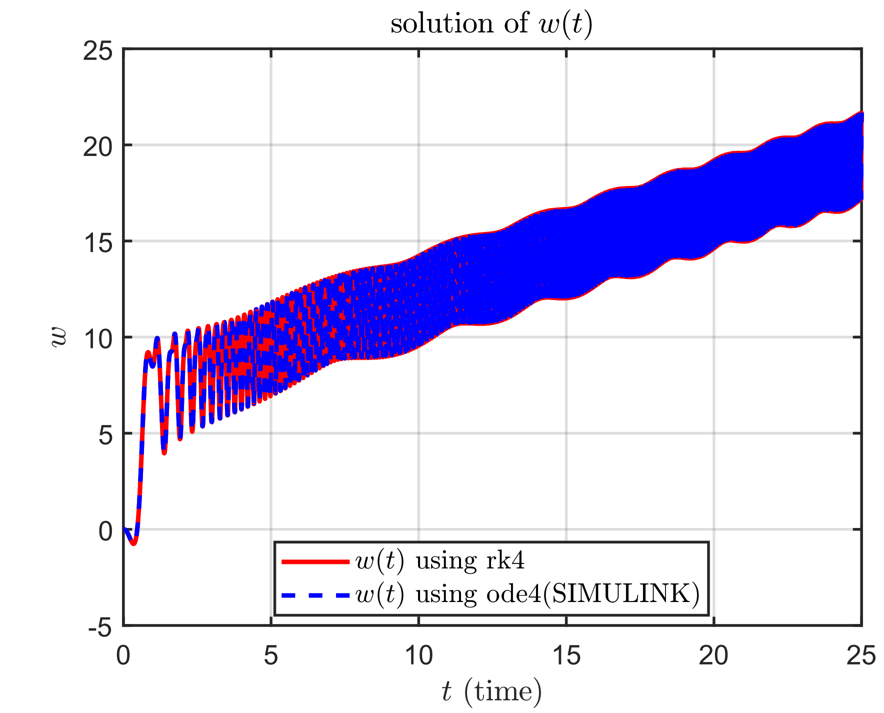


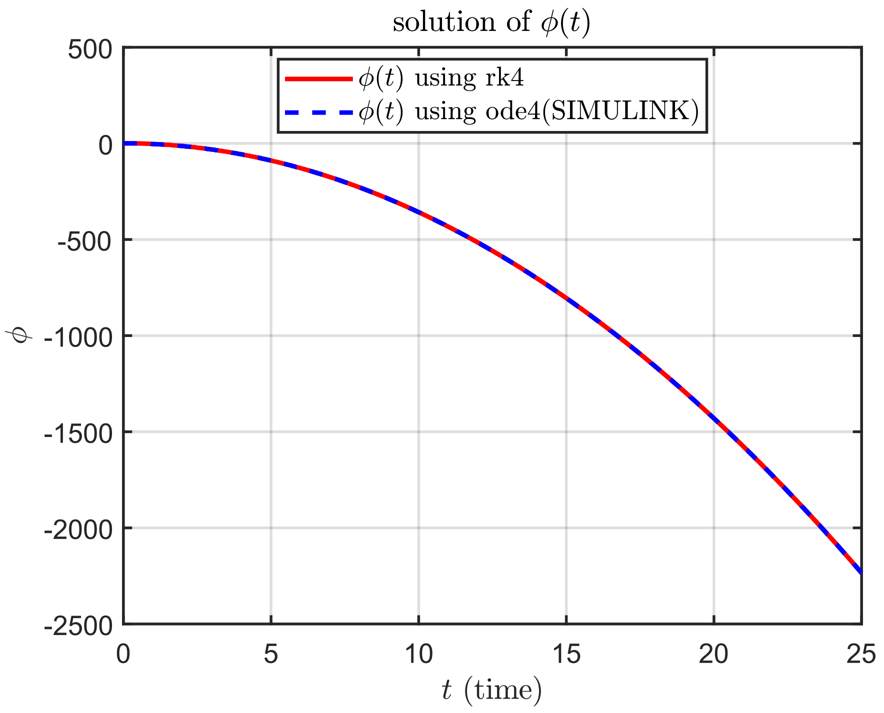

solution of $y(t)$
$y(t)$ using rk4
$y(t)$ using ode4(SIMULINK)
y
t (time)
0
50
100
150
200
250
300
5
10
15
20
25

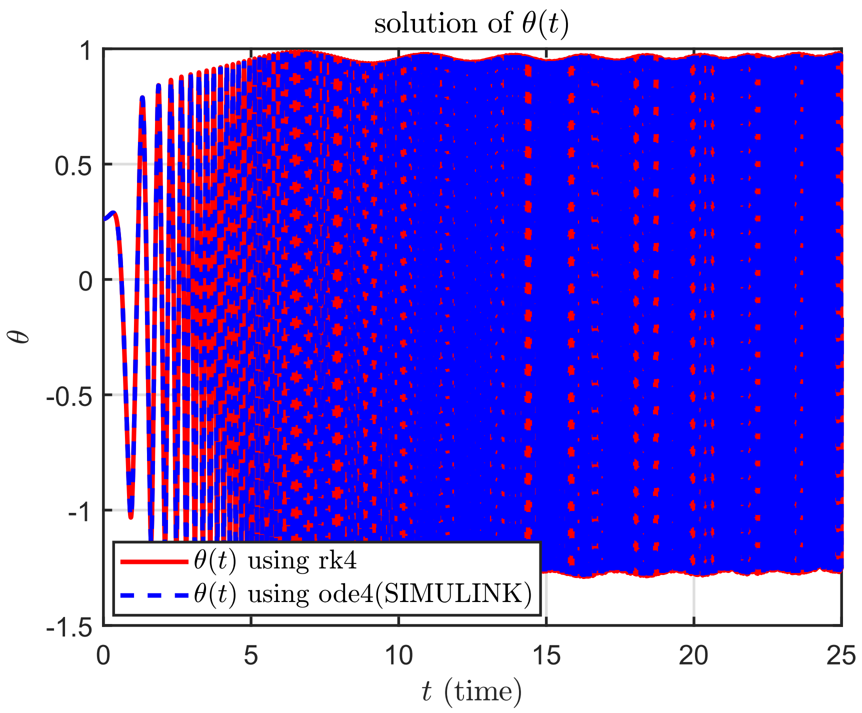

solution of $z(t)$
$z(t)$ using rk4
$z(t)$ using ode4(SIMULINK)
z
t (time)
10
5
0
-5
-10
-15
15
20
25

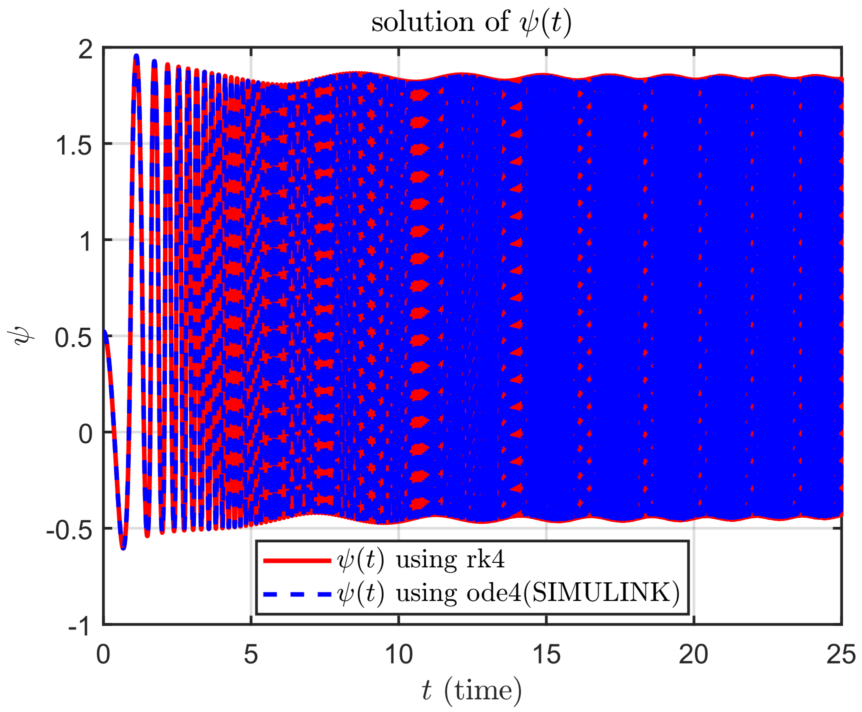

solution of $u(t)$
u
t (time)
$u(t)$ using rk4
$u(t)$ using ode4(SIMULINK)
0
5
10
15
20
25

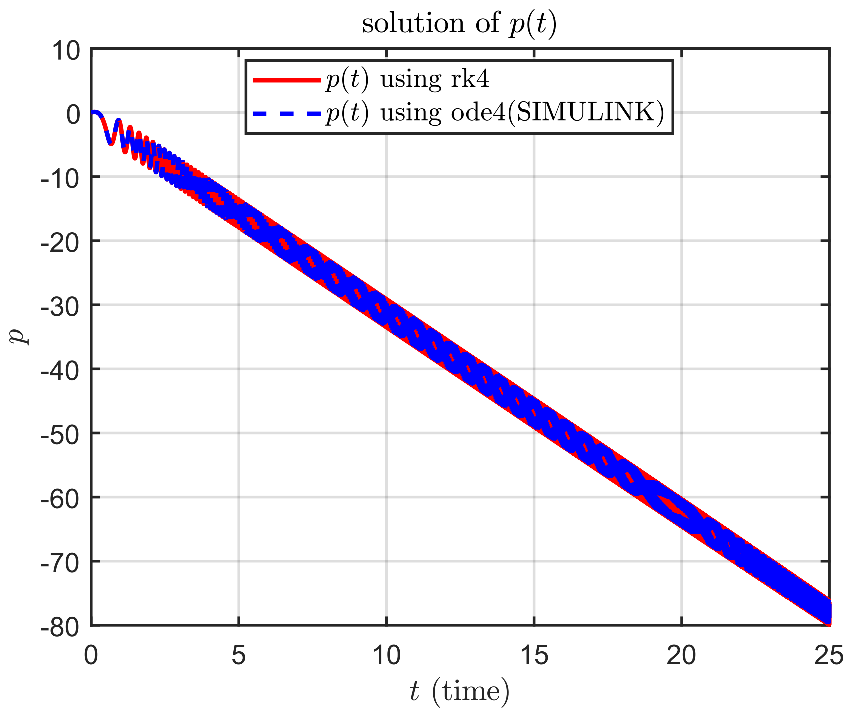

solution of $v(t)$
v
t (time)
$v(t)$ using rk4
$v(t)$ using ode4(SIMULINK)
0
2
4
6
8
10
12
14
16

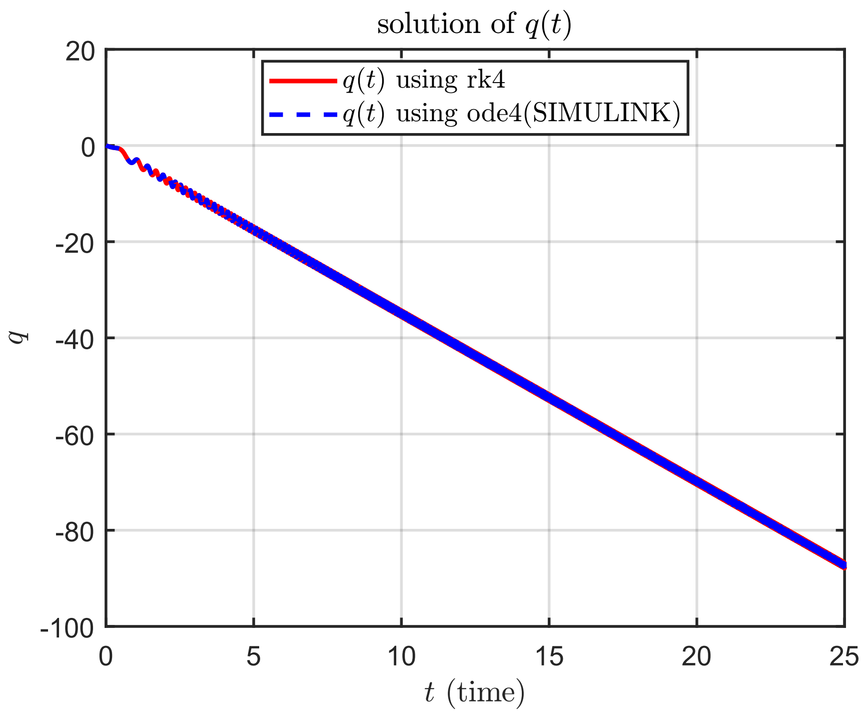

solution of $w(t)$
w
t (time)
$w(t)$ using rk4
$w(t)$ using ode4(SIMULINK)
25
20
15
10
5
0
-5
0
5
10
15
20
25

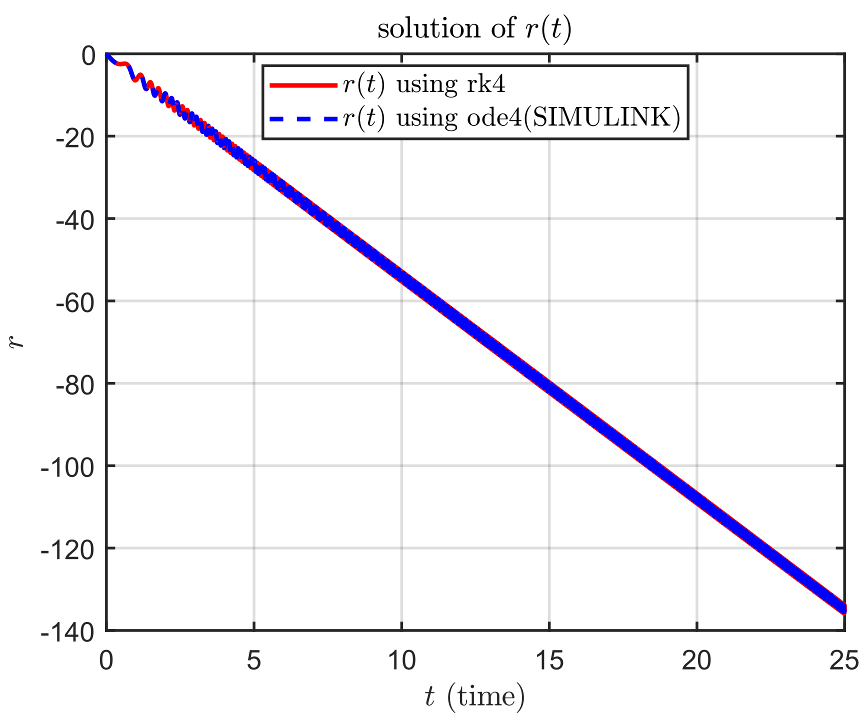

solution of $\phi(t)$
ϕ
t (time)
$\phi(t)$ using rk4
$\phi(t)$ using ode4(SIMULINK)
500
0
-500
-1000
-1500
-2000
-2500
0
5
10
15
20
25


solution of $\theta(t)$
θ
t (time)
$\theta(t)$ using rk4
$\theta(t)$ using ode4(SIMULINK)
1
0.5
0
-0.5
-1
-1.5
0
5
10
15
20
25


solution of $\psi(t)$
ψ
t (time)
$\psi(t)$ using rk4
$\psi(t)$ using ode4(SIMULINK)
2
1.5
1
0.5
0
-0.5
-1
0
5
10
15
20
25


solution of $p(t)$
$p(t)$ using rk4
$p(t)$ using ode4(SIMULINK)
p
t (time)
10
0
-10
-20
-30
-40
-50
-60
-70
-80
0
5
10
15
20
25


solution of $q(t)$
$q(t)$ using rk4
$q(t)$ using ode4(SIMULINK)
q
t (time)
20
0
-20
-40
-60
-80
-100
0
5
10
15
20
25



From the above figures it is clear that the two lines for both the code output and simulink simulation output coincide with each other demonstrating that the code results for solving RBD equations using "Runge-Kutta" 4th order method are valid.

2.3 Error Computation

This function compareErrors calculates the deviation between RK4 and Simulink solutions:

- Mean absolute error for each state variable.
- Root Mean Square Error (RMSE) to measure average deviation.
- Maximum error to identify the worst-case difference.

```
Mean Absolute Error (MAE) for each state:
    1.0e-09 *[0.0037 0.0036 0.0027 0.0050 0.0016 0.0028 0.0031 0.0012 0.0027
        0.1684 0.1931 0.0672]
Root Mean Square Error (RMSE) for each state:
    1.0e-09 *[0.0053 0.0052 0.0039 0.0071 0.0022 0.0040 0.0049 0.0019 0.0041
        0.2815 0.3240 0.1358]
Max Error for each state:
    1.0e-09 *[0.0132 0.0133 0.0099 0.0156 0.0049 0.0089 0.0193 0.0053 0.0136
        0.8145 0.9377 0.4579]
```

This page is intentionally left blank!

Appendix A

MATLAB Codes

Listing A.1: Problem1

```
clc
clearvars

% Initial conditions
t0 = 0;
Y0 = [2; 1];

% Solution parameters
tf = 20; % End time
n = 100; % Number of intervals
dt = (tf - t0) / n; % Step size
timespan = linspace(t0, tf, n+1);
Y_vec = zeros(2,n+1) ;
Y_vec(:,1) = Y0 ;

%Derivatives equation
f_vec = @(t,Y_vec) [sin(t) + cos(Y_vec(1)) + cos(Y_vec(2)) ; sin(t) + sin(Y_vec(2))];


%-------Solving the ODE using Runge-Kutta Method--------%

[t_vec_kutta,y_vec_kutta] = RK4(f_vec,timespan,Y0) ;

%-----Solving the ODE using Matlab ode45 Built in function------%

[t_vec_ode_45,y_vec_ode_45] = ode45(f_vec,timespan,Y0) ;


figure;
subplot(1,2,1);
```

```
plot(t_vec_kutta, y_vec_kutta(1,:), ’r’, ’LineWidth’, 1.5); hold on;
plot(t_vec_ode_45, y_vec_ode_45(:,1), ’b’, ’LineWidth’, 1.5); hold on;
xlabel(’t’);
ylabel(’y’);
title(’Solution␣of␣Y(1)’);
legend(’y1(t)␣using␣RK4␣Method␣’, ’y1(t)␣Using␣Matlab␣ode45␣function’);
grid on;
hold off;

subplot(1,2,2)
plot(t_vec_kutta, y_vec_kutta(2,:), ’r’, ’LineWidth’, 1.5);hold on;
plot(t_vec_ode_45, y_vec_ode_45(:,2), ’b’, ’LineWidth’, 1.5);
xlabel(’t’);
ylabel(’y’);
title(’Solution␣of␣␣Y(2)’);
legend(’y2(t)␣using␣RK4␣Method␣’, ’y2(t)␣Using␣Matlab␣ode45␣function’);
grid on;
hold off;



ERROR_1 = mean(abs((y_vec_kutta(1,:).’-y_vec_ode_45(:,1))./y_vec_ode_45(:,1)))*100

ERROR_2 = mean(abs((y_vec_kutta(2,:).’-y_vec_ode_45(:,2))./y_vec_ode_45(:,2)))*100
```

Listing A.2: RK4

```
function [t_vec,y] = RK4(states_dot,t_vec,y0)

n = length(t_vec);
y(:,1) = y0 ;
dt = t_vec(2) -t_vec(1) ;

for i = 1:n

    k1 = states_dot(t_vec(i),y(:,i)) ;
    k2 = states_dot(t_vec(i)+dt/2,y(:,i)+k1*dt/2) ;
    k3 = states_dot(t_vec(i)+dt/2,y(:,i)+k2*dt/2) ;
    k4 = states_dot(t_vec(i)+dt,y(:,i)+k3*dt) ;

    y(:,i+1) = y(:,i) + (dt/6)*(k1+2*k2+2*k3+k4) ;
end

y = y(:,1:n);
```

Listing A.3: AIRPLANE SIMULATOR PART I

```
clc;
clearvars;

%% ================= SYSTEM PARAMETERS =================
F = [2;8;3]; % External force vector (N)
M = [14;20;7]; % External moment vector (Nm)
I = [1, -2, -1; -2, 5, -4; -1, -4, 0.2]; % Inertia Matrix (kg*m^2)
m = 11; % Mass (kg)

%% ================= SIMULATION PARAMETERS =================
t0 = 0; % Initial time (s)
tf = 25; % Final time (s)
n = 20000; % Number of time steps
dt = (tf - t0) / (n - 1); % Time step size
timespan = t0:dt:tf; % Time vector

%% ================= INITIAL CONDITIONS =================
% State vector: [u, v, w, p, q, r, phi, theta, psi, x, y, z]
States_initial = [10; 2; 0; 2*pi/180; pi/180; 0; ...
                  20*pi/180; 15*pi/180; 30*pi/180; 2; 4; 7];

% Extract individual initial states for clarity
u_0 = States_initial(1); v_0 = States_initial(2); w_0 = States_initial(3);
p_0 = States_initial(4); q_0 = States_initial(5); r_0 = States_initial(6);
phi_0 = States_initial(7); theta_0 = States_initial(8); epsi_0 = States_initial(9);
x_0 = States_initial(10); y_0 = States_initial(11); z_0 = States_initial(12);

%% ================= SOLVE USING RUNGE-KUTTA 4 =================
States_dot_fn = @get_states_dot;
[t_vec_kutta, states_vec_kutta] = RK4(States_dot_fn, timespan, States_initial);

%% ================= SOLVE USING SIMULINK =================
sim('sim_6_dof.slx');
y_vec_simulink = [uvw_out, anglerates_out, angles_out, xyz_out];

%% ================= PLOT RESULTS =================
plotStateComparisons(t_vec_kutta, states_vec_kutta, tout, y_vec_simulink);

%% ================= ERROR ANALYSIS =================
[abs_error, rmse, max_error] = compare_errors(t_vec_kutta, states_vec_kutta, tout',
    y_vec_simulink');
```

Listing A.4: getStatesDot

```
function states_dot = get_states_dot(t,states_vec)

F = [2;8;3] ;

M = [14;20;7] ;

I = [1,-2,-1;-2,5,-4;-1,-4,0.2];

m = 11;

u = states_vec(1);
v= states_vec(2);
w= states_vec(3);


p= states_vec(4);
q= states_vec(5);
r= states_vec(6);


phi= states_vec(7);
ceta= states_vec(8);
psi= states_vec(9);
eul = [psi ceta phi] ;

X= states_vec(10);
Y= states_vec(11);
Z= states_vec(12);


states_dot (1:3,1) = (1/m)*F-cross([p;q;r],[u;v;w]) ;

states_dot (4:6,1) = inv(I)*[M-cross([p;q;r],I*[p;q;r])] ;

states_dot (7:9,1) = [1, sin(phi)*tan(ceta), cos(phi)*tan(ceta) ; 0, cos(phi), -sin(phi)
    ; 0, sin(phi)/cos(ceta), cos(phi)/cos(ceta)] * [p;q;r];

states_dot (10:12,1) = eul2rotm(eul,"ZYX")*[u;v;w];




states_dot = states_dot (1:12,1);
```

```
end
```

Listing A.5: plotStateComparisons

```
function plotStateComparisons(t_vec_kutta, states_vec_kutta, t_vec_ode_45, y_vec_ode_45)
    % Define state labels in lowercase (LaTeX format) and filenames
    state_labels_latex = {'u', 'v', 'w', 'p', 'q', 'r', '\phi', '\theta', '\psi', 'x', 'y', 'z'};
    state_labels_filename = {'u', 'v', 'w', 'p', 'q', 'r', 'phi', 'theta', 'psi', 'x', 'y', 'z'};

    % Define colors and line styles
    rk4_style = {'r-', 'LineWidth', 2}; % Red solid line for RK4
    ode45_style = {'b--', 'LineWidth', 2}; % Blue dashed line for ode45

    % Loop through all 12 states
    for i = 1:12
        figure;

        % Plot RK4 and ode45 solutions
        plot(t_vec_kutta, states_vec_kutta(i, :), rk4_style{:}); hold on;
        plot(t_vec_ode_45, y_vec_ode_45(:, i), ode45_style{:});

        % Set labels with LaTeX in lowercase
        xlabel('$t$ (time)', 'Interpreter', 'latex', 'FontSize', 14);
        ylabel(['$', state_labels_latex{i}, '$'], 'Interpreter', 'latex', 'FontSize', 14);
        title(['solution of $', state_labels_latex{i}, '(t)$'], 'Interpreter', 'latex', 'FontSize', 16);

        % Add legend
        legend({['$', state_labels_latex{i}, '(t)$ using rk4'], ...
              ['$', state_labels_latex{i}, '(t)$ using ode4(SIMULINK)']}, ...
              'Interpreter', 'latex', 'FontSize', 12, 'Location', 'Best');

        % Beautify grid and axis
        grid on;
        set(gca, 'FontSize', 12, 'LineWidth', 1.2, 'Box', 'on');

        % Save the figure as an SVG file with lowercase names
        filename = sprintf('solution_%s.svg', state_labels_filename{i});
        saveas(gcf, filename, 'svg');

        hold off;
    end
```

```
end
```

Listing A.6: compareErrors

```
function [abs_error, rmse, max_error] = compare_errors(t_rk4, y_rk4, t_ode45, y_ode45)

    % Compute Absolute Error
    abs_error = abs(y_rk4 - y_ode45);

    % Compute Mean Absolute Error (MAE) for each state separately
    mae = mean(abs_error, 2); % Mean across time steps (row-wise for each state)
    % Compute RMSE (for each state separately)
    rmse = sqrt(mean(abs_error.^2, 2)); % Row-wise RMSE (each row = one state)

    % Compute Max Error (for each state separately)
    max_error = max(abs_error, [], 2); % Max across time steps

    % Display Results
    disp('Mean␣Absolute␣Error␣(MAE)␣for␣each␣state:');
    disp(mae);
    disp('Root␣Mean␣Square␣Error␣(RMSE)␣for␣each␣state:');
    disp(rmse);
    disp('Max␣Error␣for␣each␣state:');
    disp(max_error);
end
```

Bibliography

[1] J. Blakelock, *Automatic Control of Aircraft and Missiles*, ser. A Wiley-Interscience publication. Wiley, 1991. [Online]. Available: https://books.google.com.eg/books?id=ubcczZUDCsMC

[2] B. Stevens, F. Lewis, and E. Johnson, *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems.* Wiley, 2015. [Online]. Available: https://books.google.com.eg/books?id=boybCgAAQBAJ

[3] M. Sadraey, *Unmanned Aircraft Design Techniques*, ser. Aerospace Series. John Wiley & Sons Canada, Limited, 2020. [Online]. Available: https://books.google.com.eg/books?id=IX9IzQEACAAJ