Dynamic Modeling and PID Control of a DC Motor Using MATLAB/Simulink

1. Introduction

This project focuses on modeling the electromechanical dynamics of a DC motor in MATLAB Simulink, analyzing its open-loop behavior, and designing a PID controller to regulate the motor speed. Additionally, numerical solver behavior (ode45 vs. ode15s) is compared.

2. Mathematical Model of the DC Motor

Electrical Equation:
$L * di/dt = V - R*i - Ke*\omega$
Mechanical Equation:
$J * d\omega/dt = Kt*i - B*\omega - TL$
Where V is applied voltage, i is current, ω is angular speed, R is resistance, L is inductance, Ke is back EMF constant, Kt is torque constant, J is inertia, B is damping, and TL is load torque.

3. Simulink Implementation

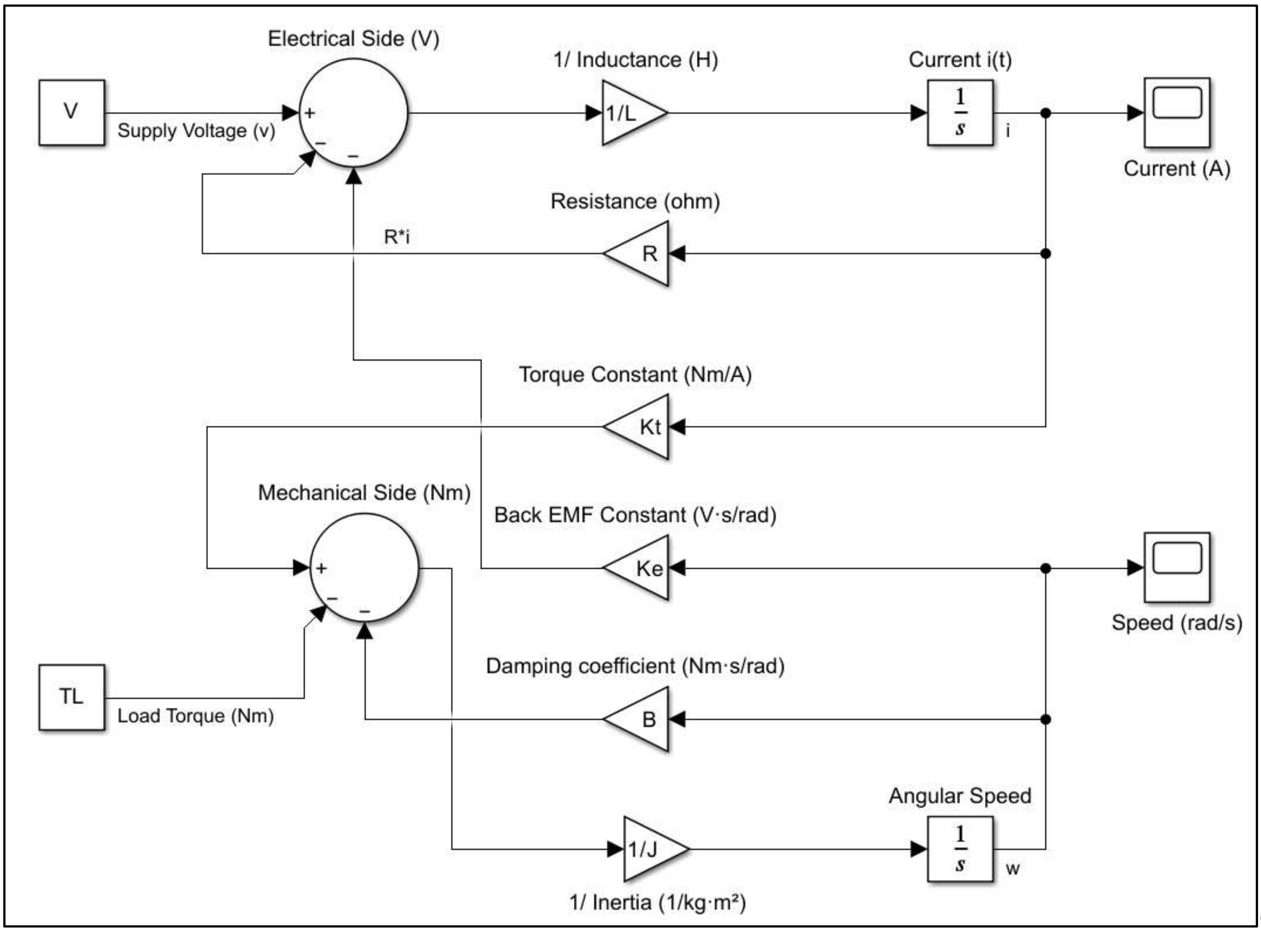
The model consists of electrical and mechanical subsystems connected through torque and back-EMF. Sum, Gain, Integrator, and Scope blocks form the dynamic structure.

4. Open-Loop Results

Initial Setup to verify that the mathematical model is working accurately and validate it against numerical calculations. Open-loop results show current saturating at V/R = 12 A and speed stabilizing near ~0.2 rad/s, consistent with theoretical steady-state calculations. Using values;

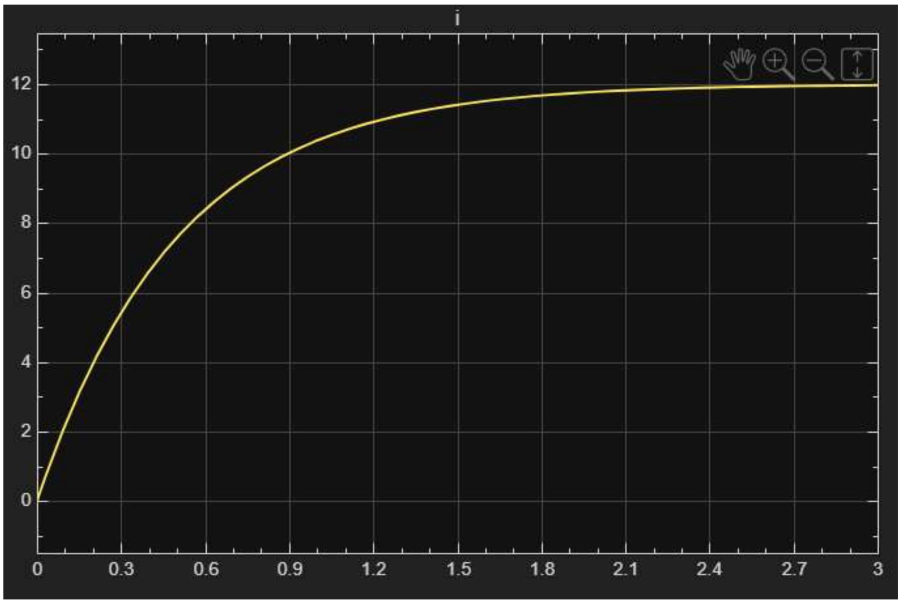
```
R = 1; % Resistance (ohm)
L = 0.5; % Inductance (H)
Ke = 0.01; % Back EMF constant
Kt = 0.01; % Torque constant
J = 0.01; % Inertia (kg.m^2)
B = 0.1; % Damping coefficient
TL = 0.1; % Load Torque
V = 12; % Supply voltage
```

MATLAB MODEL



Current

Curve i(t)



This is **classic RL (inductor + resistor) behaviour**:

- At t = 0, current starts at 0.
- It rises exponentially.

- It approaches a steady value given by:

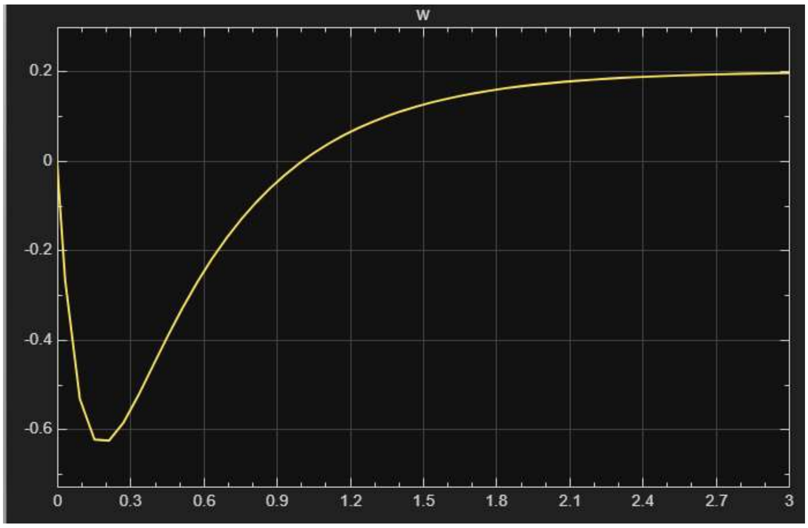
$$i_{ss} = \frac{V}{R}$$

- Since **V = 12, R = 1**,

$$i_{ss} = 12$$

- Your curve is correctly approaching 12 A.

Speed Curve ω(rad/s)



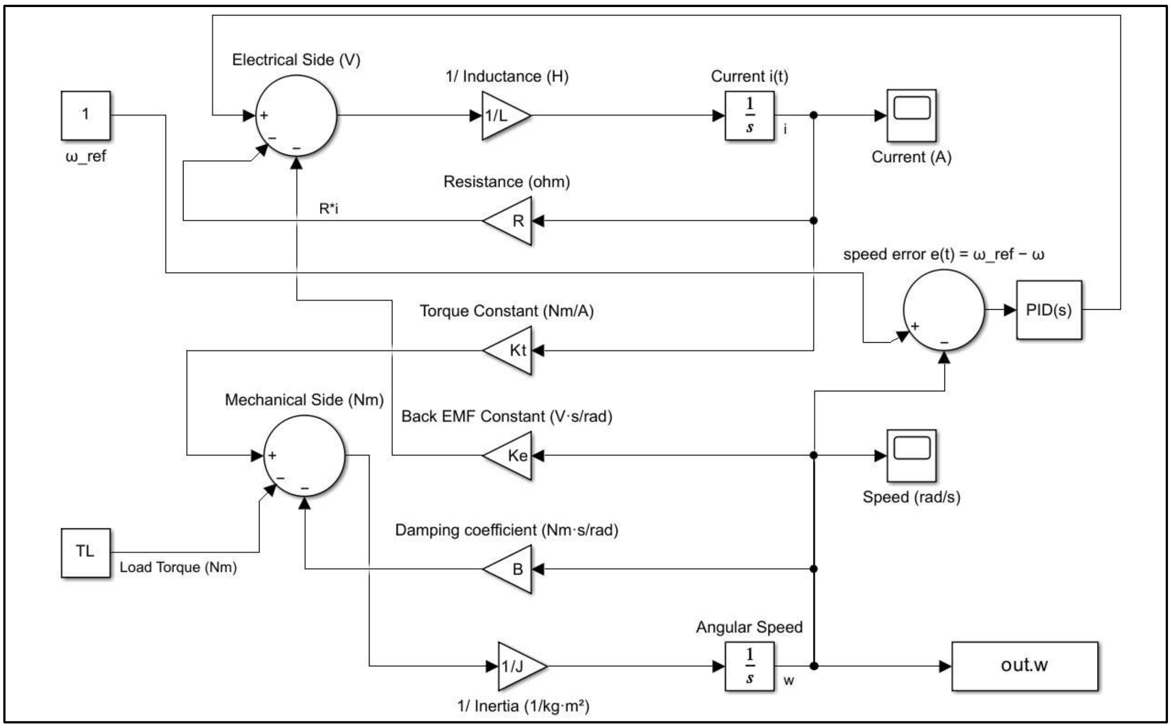
$$\omega_{ss} = \frac{K_t i - T_L}{B}$$

$$\omega_{ss} = \frac{0.12 - 0.1}{0.1} = 0.2 \text{ rad/s}$$

Plot approaches **0.2 rad/s exactly**

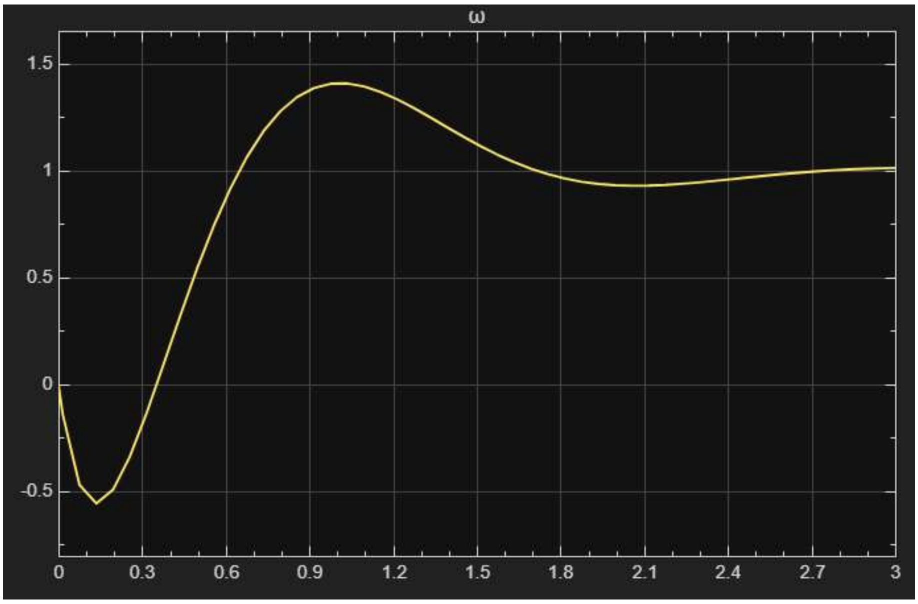
5. PID Speed Control Design

Turing the constant velocity into a speed reference using the error block (ω_ref − ω) and to control the motor using PID Turning.



1st Iteration

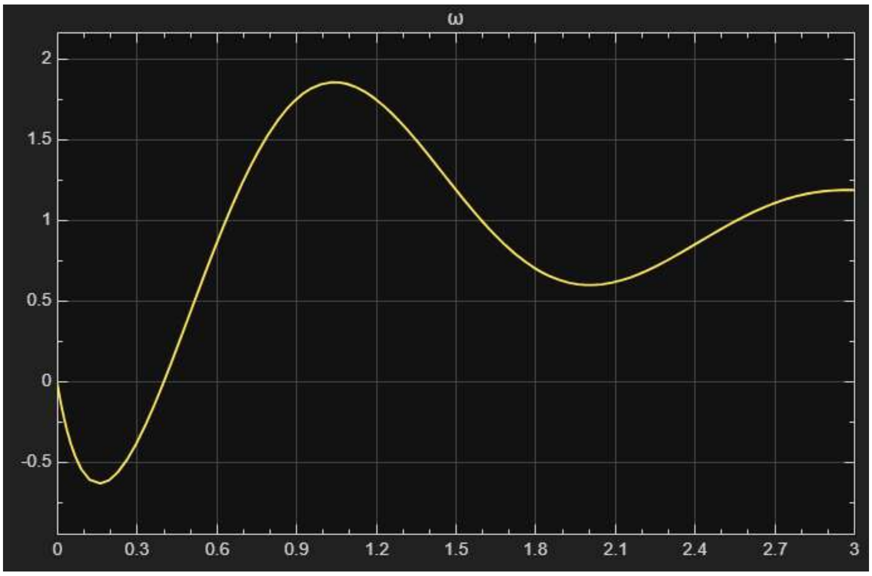
P = 10 I = 50 D = 0



Reduce Overshoot (Reduce P). Faster Push to Settle (Increase D a Little)

2nd Iteration

P = 6 I = 40 D = 0.1

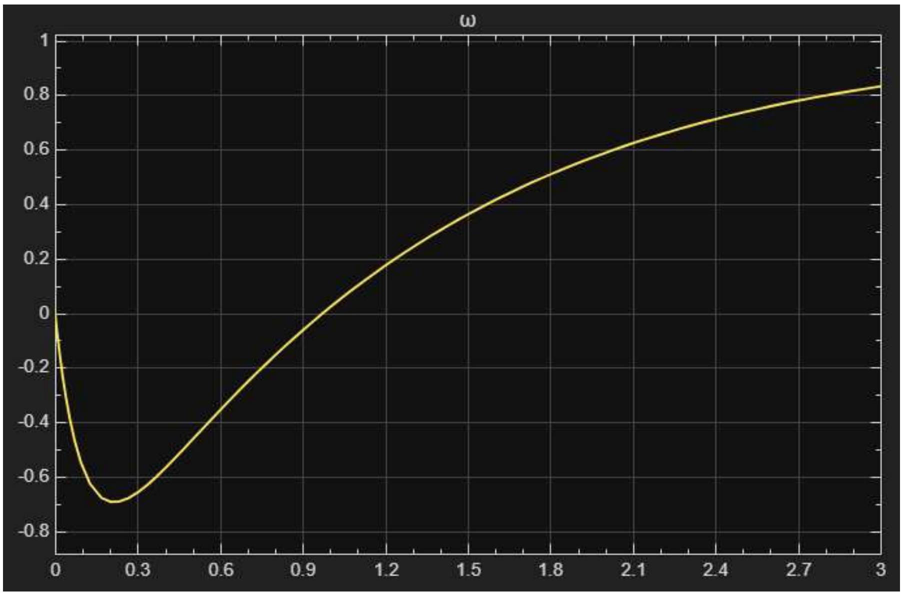


Peak ≈ **1.8** for a reference of 1 → **~80% overshoot**

- It dips and comes back up again → clear **oscillation**
- At t = 3 s it still hasn't fully settled → **integral too strong**, P still a bit high
- **Lower P** → less "push", reduces overshoot
- **Much smaller I** → reduces integral windup & oscillation
- **Slightly higher D** → adds damping, smooths the peak

3rd Iteration

P = 4 I = 8 D = 0.2



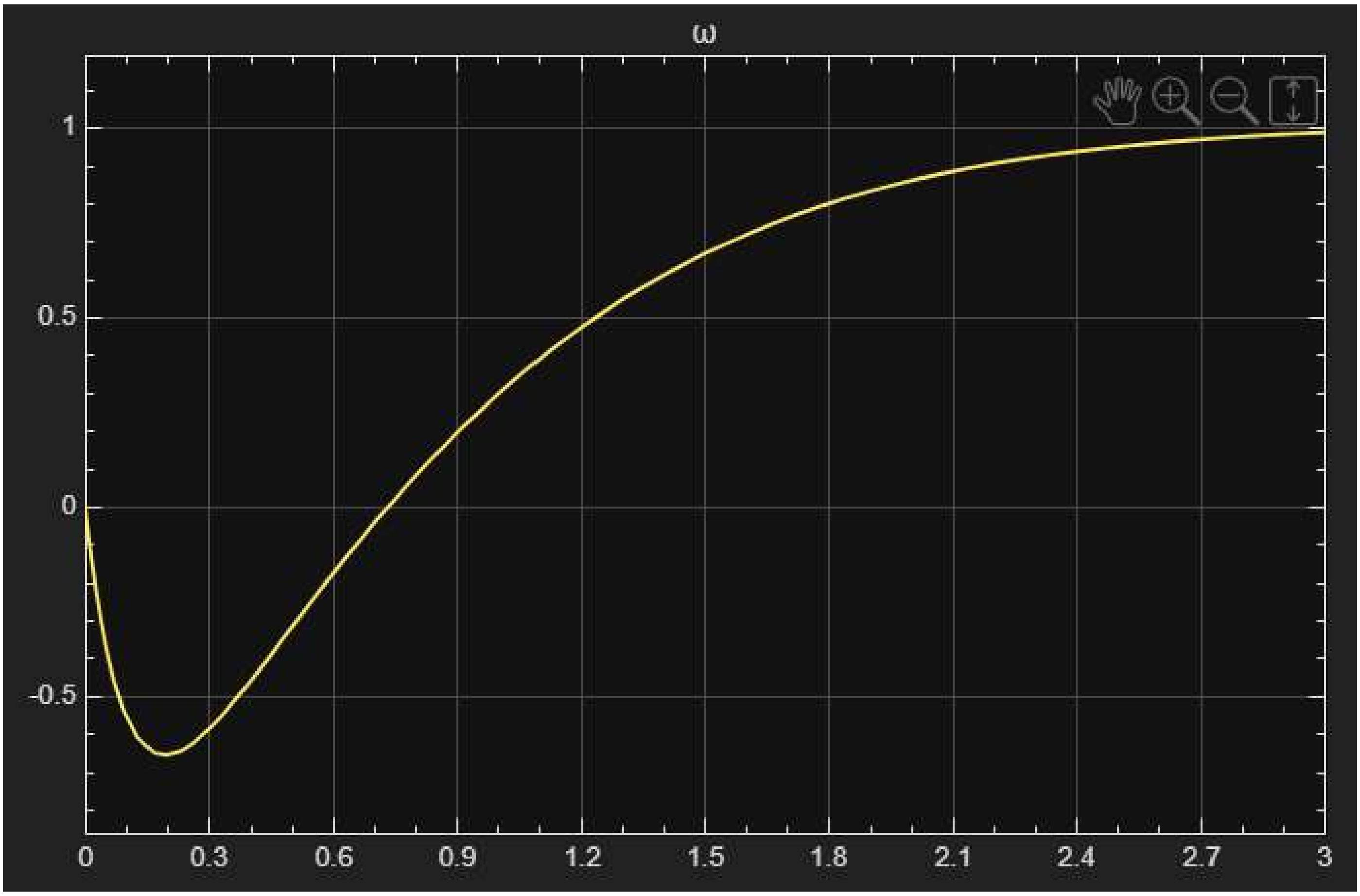
Increase Stop time to 10 s and run. Faster climb, maybe a tiny overshoot (<10%) or just a smooth approach. Settling around 1 rad/s

Final Iteration

Manual PID tuning was used. Final gains:

P = 5 I = 12 D = 0.2

These gains produced smooth, stable tracking of the 1 rad/s reference.



6. Solver Comparison

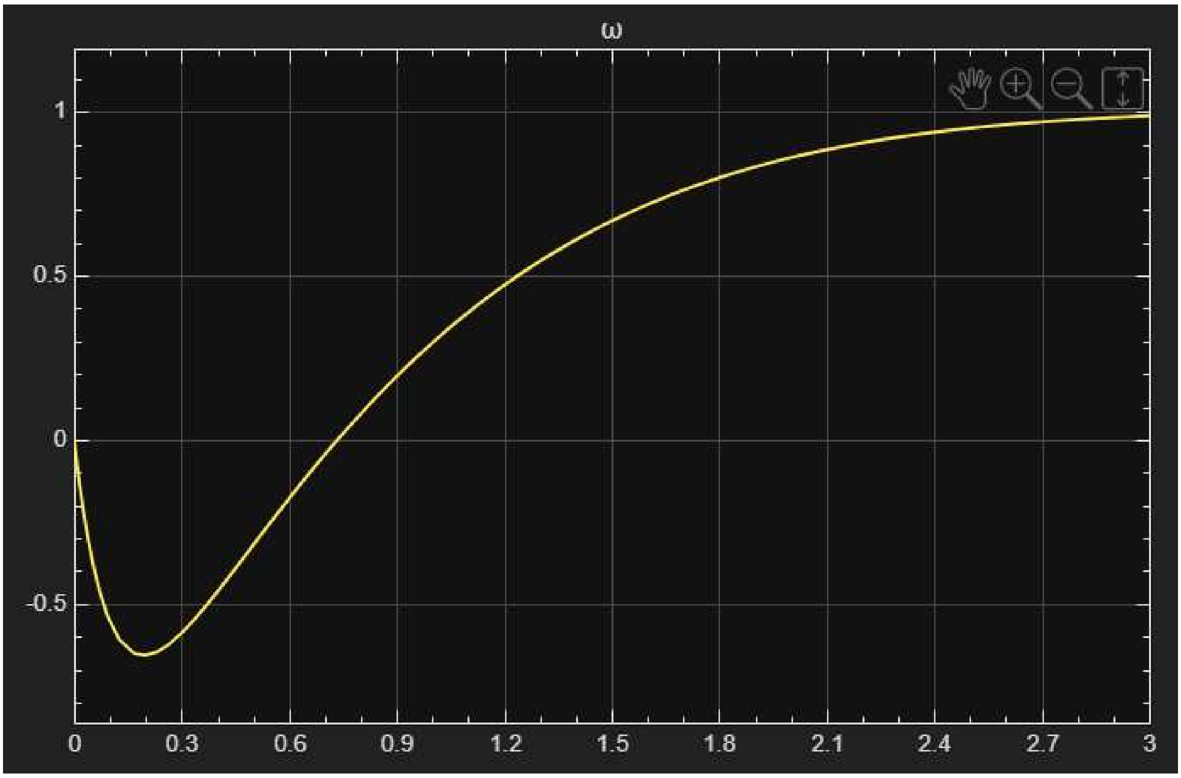
Compare ode4 (Dormand–Prince) vs ode15s (stiff/NDF).

To extract various response metrics using the ‘To Workspace’ block

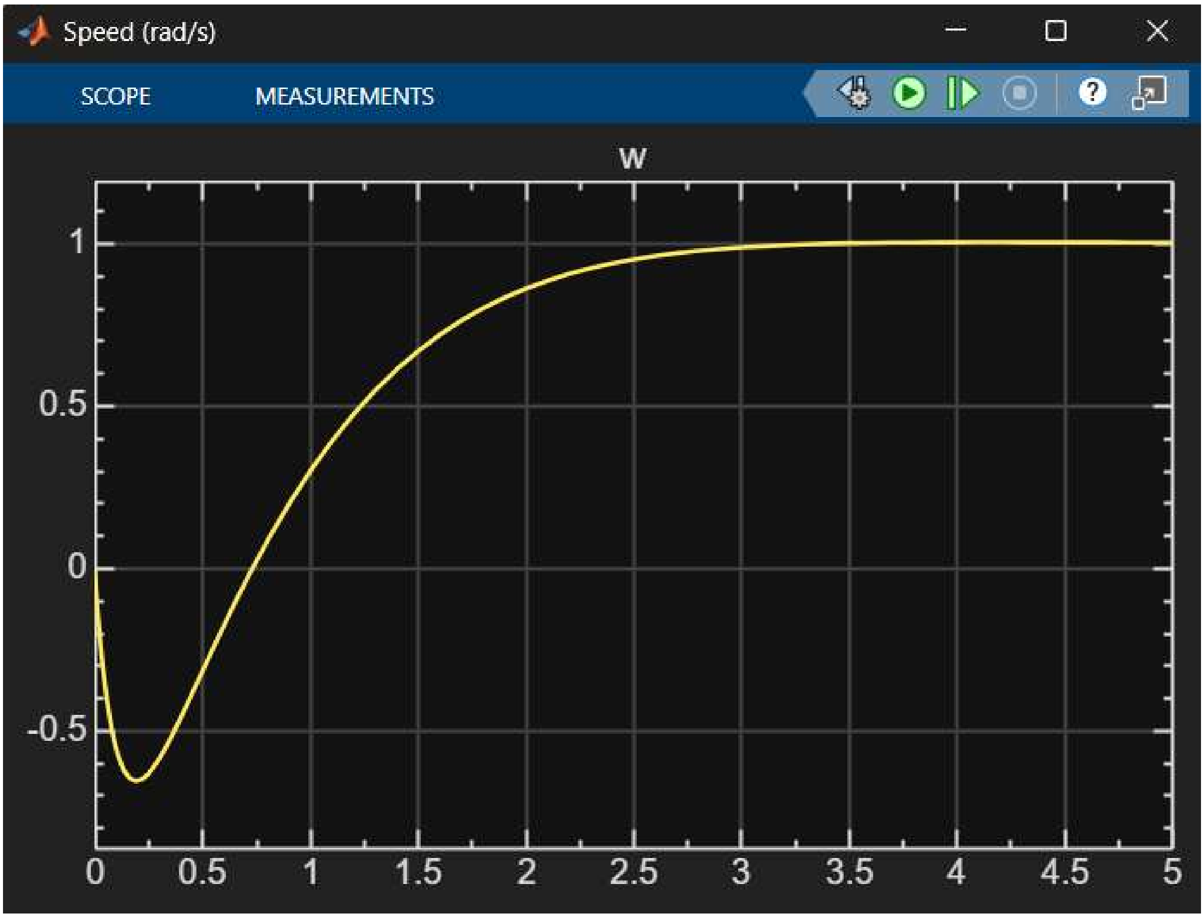
For ode45 : info_ode45 = stepinfo(out.w.signals.values, out.w.time);

Then switch solver to ode15s : nfo_ode15s = stepinfo(out.w.signals.values, out.w.time);

Ode45 Curve



Ode15s Curve



Comparison of step response metrics:

Metric	ode45	ode15s
Rise Time (s)	1.3031	1.3687
Transient Time (s)	2.5392	2.6987
Settling Time (s)	2.6784	2.8976
Settling Min	0.8923	0.9057
Settling Max	0.9868	1.0030
Overshoot (%)	0	0.1195
Undershoot (%)	66.5680	65.5590
Peak Value	0.9868	1.0030
Peak Time (s)	3	4.2058

7. Conclusion

The closed-loop model demonstrates strong performance with minimal overshoot and smooth settling. ode45 performed slightly faster, while ode15s provided marginally smoother convergence. Both solvers are suitable for this moderately stiff system.